BHBIA RFP ASSETS

A Technical Audit and subsequent workshop report and recommendations for the rebuild of the BHBIA website

Adam Prentice

Darkish Ltd.

Table of Contents

1. Executive Summary

1.1 Overview

This report summarises the outcomes from the Technical Audit and subsequent Requirements Gathering Workshop conducted for the BHBIA website. The audit assessed the current state of the system, identifying critical issues in areas such as technology stack, security, performance, and administrative workflows. The workshop further clarified the business requirements, highlighting areas needing significant improvement.

1.2 Key Issues Identified

Outdated & Unsupported Technologies
The current system is built on technologies no longer supported, which pose severe security risks and compatibility challenges, making maintenance difficult.

Security Vulnerabilities
Significant vulnerabilities were identified, including SQL injection risks and Cross-Site Scripting (XSS). Additionally, essential security headers (e.g., CSP) are missing, increasing susceptibility to attacks.

Administrative Complexity
Current administrative processes are predominantly manual, complex, and time-consuming, particularly regarding payments, event management, and reporting. The admin area's performance is notably poor during busy periods.

Performance & User Experience Issues
General site performance suffers from inefficient caching, slow load times (especially during renewal periods), and poor content optimisation (e.g., large images, layout shifts, render-blocking scripts).

Limited Payment & Financial Automation
Financial processes are fragmented, with manual invoicing, limited automation for debt collection, no integrated VAT handling, and no direct export functionality for accounting software (Sage).

Reporting Inefficiencies
The reporting system relies heavily on manually created templates and lacks flexibility. Reports cannot cover multiple database tables simultaneously, making data extraction and auditing cumbersome. Historical data is not shown on the site, and so reports are utilised to manually store the data for future use.

Administrative Complexity & Technical Debt
The administration area lacks efficient workflows, suffers from duplication of content, and is heavily reliant on manual intervention for updates, reporting, and payment management. Document management is inefficient and causes issues when needing to reuse documents and images across the site.

1.3 Scope of the Audit

The audit encompassed a thorough review of the provided codebase, including:

- Application Code: Evaluation of coding standards, maintainability, scalability, and security.
- Architecture & Design: Assessment of the overall system design, modularity, and adherence to best practices.
- Database Design: Review of schema structure, indexing, query efficiency, and data integrity.
- System Performance: Identification of bottlenecks and areas for optimization.
- Security Considerations: Examination of potential vulnerabilities and adherence to security best practices.
- Website Functionality: A hands-on review of the live website to assess usability, responsiveness, and any observable performance or security issues.
- Admin Panel Review: Analysis of the administrative interface to identify inefficiencies, usability concerns, and areas for workflow improvement.

1.4 Recommendation

A full rebuild of the website is recommended to address these issues effectively. The new build should employ modern technologies, significantly enhancing security, performance, and usability. Centralising payment management through Stripe and simplifying content administration via structured CMS templates are crucial recommendations.

2. High-Level Business Objectives

The following objectives have been defined to guide the redevelopment of the website:

Streamlined User Experience (UX)
Improve overall user journeys, simplify registration and renewal processes, and facilitate better content discovery and site interaction.

Simplified Administration
Reduce manual administration tasks through improved workflows, intuitive interfaces, and automation.

Enhanced Security and Performance
Address identified security vulnerabilities, improve site performance, especially during busy periods (renewal times), and ensure best practices in data security and compliance.

Improved Payment Automation and Financial Handling
Centralise and automate invoicing, payment handling, debtor management, and integrate with accounting software through Stripe.

Efficient Content Management and Administration
Implement a CMS with structured templates to simplify content management, improve ease of use for administrators, and reduce complexity.

Improved Marketing and Engagement Strategies
Utilising Customer Segmentation, recommendations and improved CTAs to encourage continued engagement with the site

3. Existing System

3.1 Description

The current system is built using Statamic, a flat-file content management system (CMS) that runs on the Laravel framework, utilizing the PHP programming language. A MySQL database is used for data storage, and Redis is used for caching and session management.

While the exact age of the system is unconfirmed, several indicators suggest that it has been in operation for at least seven years without significant maintenance. This conclusion is based on observed legacy coding practices, outdated framework versions, and dependencies that are no longer supported. The system contains data dating back to 2009, but it is unclear whether this data was migrated from a previous system or originated within the current implementation. Therefore, this data cannot be used as a definitive indicator of the system's age.

The codebase is large and highly customized, extending beyond a typical CMS implementation. Rather than being a standard Statamic-based website, it is more accurately described as a custom web application built on top of Statamic, incorporating significant bespoke functionality.

It has been confirmed by the previous developer that the site has not been updated due to compatibility issues with newer versions of Statamic and Laravel. Updating to a more recent version would require a substantial rewrite of the system's core functionality, making incremental updates impractical. As a result, the application is currently running on end-of-life software versions, posing significant risks:

- Security vulnerabilities: Older versions of Laravel, Statamic, and PHP no longer receive security patches, making the system susceptible to exploits.
- Compatibility issues: Newer third-party dependencies, libraries, and hosting environments may not support outdated versions, limiting maintainability.
- Performance limitations: Modern PHP and Laravel versions provide significant performance improvements that are unavailable in the current system.
- Long-term sustainability: As technology continues to evolve, reliance on outdated software will make future upgrades and integrations increasingly difficult.

A more detailed analysis of these risks and their implications is provided in the Audit Findings section under Dependencies.

3.2 Technical Stack

The following technologies are used in the current system:

Category	Technology	Notes & Considerations
Framework & CMS	Laravel / Statamic V2	Statamic V2 is built on Laravel 5.x, both of which are outdated and no longer actively supported. Significant custom functionality has been built on top of the CMS. Upgrading to a

		newer version would require substantial redevelopment.
Programming Language	PHP 7.1	PHP 7.1 reached end-of-life in December 2019, meaning it no longer receives security updates. Running unsupported versions presents security risks and limits compatibility with modern libraries.
Database	MySQL 5.7	End-of-life as of October 2023. Future hosting environments may not support it, and performance/security optimizations in newer MySQL versions are unavailable.
Templating Engine	Antlers	Statamic's proprietary templating engine. Offers some benefits in terms of flexibility but can limit portability if migrating away from Statamic.
Containerization	Docker	Used for local development or deployment. If properly configured, it simplifies environment consistency but adds an extra layer of complexity in maintenance.
Caching & Sessions	Redis	Used for caching and session management. This is an appropriate choice for improving performance but requires ongoing monitoring to prevent excessive memory usage.
Hosting Provider	Digital Ocean	The application is currently hosted on Digital Ocean, a flexible but self-managed cloud provider. While cost-effective, it lacks the managed services of platforms like AWS or Azure, requiring more administrative oversight.
Asset Storage	Git (Version Control)	Due to Statamic's flat-file nature, assets are stored in Git and on the server rather than in cloud storage. This can lead to scalability issues as asset volumes grow.
Deployment & CI/CD	Laravel Forge	Used for managing deployments. Developer documentation suggests that deployments are automated, reducing the risk of manual errors. However, it is unclear if rollback strategies or environment-specific configurations are implemented.
Live Changes Sync	Statamic Plugin	Used to sync live environment changes back to the repository. This suggests some workflows involve direct changes to production, which can introduce risks if not properly managed.
Transactional Email	Mandrill	Used for sending system-generated emails. Mandrill is an add-on for Mailchimp and may require a Mailchimp account for continued use. Alternative providers such as Postmark or Amazon SES could offer more flexibility.
Payment Processing	Stripe	Used for processing online payments. Stripe is a robust and widely used solution, but ongoing updates to API

		versions should be monitored to avoid compatibility issues.
Marketing Email	Campaign Monitor	Used for marketing email campaigns. Integration should be reviewed to ensure compliance with GDPR and other relevant data protection regulations.

3.3 Main Sections & Features

The system consists of several key sections, each providing distinct functionalities tailored to administrators, members, and the public. These sections range from content management (CMS) features to e-commerce capabilities, event management, training modules, and membership-based access.

Administration & CMS

The platform includes an administration section that is accessible only to users with Administrator status. This section provides a range of management features, including:

- Content Management System (CMS): Allows administrators to create and manage collections such as Events, Online Training, and creating and amending individual pages.
- User & Organization Management: Administrators can manage members and organizations, granting permissions and overseeing memberships.
- Report Generation: A built-in reporting system enables data extraction, though it is limited in scope:
    - Reports can only be generated on a single database table at a time.
    - Reports require the creation of a template before generating a report.
    - Due to this workflow, many templates have accumulated, often with only slight variations between them.

CMS-Driven Pages

The website includes multiple pages that are managed through the CMS:

- Flexible Content Pages: Most pages (e.g., About, Resources, News) allow for flexible page-building through a CMS-based editor.
- Custom Templates: Some pages utilize predefined templates, such as Committee & Board Members and BOBI Awards
- Collection-Based Pages: Sections such as Events, Online Training, and News are dynamically built from content collections stored within the CMS.

Online Training

The system provides an online training platform that allows members to purchase and complete courses. Training modules are fully managed within the system using a custom implementation.

- Course Structure: Each training module consists of individually created slides and questions, which are then associated with a specific course.
- Purchasing & Access:

    - Once purchased, training content remains accessible until it is manually deleted from the CMS.
    - If a course is marked as “Draft” or hidden, it is still available to users who previously purchased it.
- Lack of Historical Tracking:
    - No version history is maintained for courses or questions.
    - If an administrator updates a course or modifies quiz questions, all users—including those who have already completed the training—will see the latest version. This could create inconsistencies in certification records.
- Certification Expiry: Certifications expire annually on October 31st.

Events

The platform allows members to purchase and book onto events. Events include a range of customizable settings:

- Event Customization: Events can be configured with custom pricing, delegate limits (for different member categories such as Agency and Industry), and multiple ticket types.
- Calendar Integration: The system can generate an ICS file for users to download, allowing them to add the event to their calendar.
- Email Customization: Administrators can define the email content that gets sent to registered delegates.
- Webinar Access:
    - A webinar link must be included in the email content for attendees to access online events.
    - There is no direct mechanism for accessing a webinar through the website itself.
- Event Duplication:
    - While template-based or recurring events are not supported, an existing event can be duplicated to create a new one.

BOBI Awards

The BOBI Awards section allows members to submit applications for awards. The system provides:

- A publicly accessible list of available awards.
- The ability to sponsor awards.
- Submission Process:
    - Applications are handled externally through SmartSurvey rather than within the system.
    - Previous winners are manually added to the website each year.

Job Board

The system includes a job board feature where members can post job vacancies. However, usage appears to be low. Only 6 job listings were submitted in 2024 and 10 in 2023. It is unclear if the restrictions on who can post jobs or usability limitations are the cause of the low engagements. Job Boards are hidden in a dropdown on the menu, and so it’s likely it’s not frequently accessed by users.

Members Directory

The platform includes a member's directory that lists member companies. Paid members gain access to contact details of companies, while non-members have restricted visibility.

Member Profile

Each member has a profile page, which serves as a central hub for accessing their purchased content and historical activity. From here, the member can access their stored content, such as Training they've completed, Events they've attended and Job's they've submitted.

4. Technical Audit Findings

4.1 Code Quality

The overall code quality of the application presents several challenges, primarily due to legacy practices, inconsistent standards, and a lack of maintainability. While the core structure of the system is functional, the presence of hardcoded values, duplicated logic, inconsistent naming conventions, and outdated models contributes to technical debt.

One of the most significant issues is the use of hardcoded values throughout the application. This is particularly evident in the Access Middleware, where URLs are embedded directly within the code, reducing flexibility and increasing the difficulty of making environment-based configuration changes. Hardcoded email subject lines also limit adaptability, making it challenging to modify messaging dynamically without altering the codebase. To enhance maintainability, these values should be extracted into configurable parameters within environment files, a dedicated settings module or directly within the CMS, depending on the amount of flexibility required.

The organization of files follows expected conventions for Laravel and Statamic, with core functionalities such as the CMS, Theme, and custom implementations in add-on's. However, some areas lack a clear organizational pattern, making it difficult for developers to navigate the system efficiently. Additionally, there is duplication of code across multiple models. For example, similar logic for calculating payment types, costs, and Stripe URLs appears in multiple models, including JobVacancy, EventBooking, AdvertisingRequest, and CoursePurchase. A more structured approach using base model classes would help to reduce redundant code and improve maintainability.

There are also inconsistencies in date handling, where different database entities store and format dates in varying ways. Naming conventions also lack uniformity, with some fields using camelCase (e.g., createdBy, submittedBy) while others follow snake_case (created_by, submitted_by). These inconsistencies make it harder to maintain the codebase and introduce unnecessary complexity when querying the database. A more standardized approach to naming and date formatting should be enforced across the application.

A mixture of legacy and modern models is another concern, as older parts of the system still follow outdated Laravel conventions while newer components adhere to modern best practices. This inconsistency creates confusion for developers working across different parts of the application and complicates maintenance efforts.

4.2 Reporting System

The application's reporting system is well-structured, with reports inheriting from a base class and providing a standardized mechanism for data extraction. However, reports rely on hardcoded SQL statements, which, while functional, reduce maintainability compared to an ORM-based approach. Although this issue is not user-facing, transitioning to using the ORM would provide greater flexibility and security.

Another issue is the lack of data versioning in reports. Currently, reports do not maintain historical snapshots, meaning that once data changes, previous reports become inaccurate and irreproducible. This could pose challenges for auditing and long-term reporting needs. Implementing archived reports or a versioned reporting system would provide better data integrity and traceability.

Additionally, error handling within reports is minimal. There is little validation on database joins, meaning that incorrectly structured queries may fail silently rather than returning meaningful errors. Similarly, the report formatters do not include robust error handling, making debugging difficult if issues arise. Enhancing validation and logging within the reporting system would improve reliability.

4.3 Error Handling & Logging

The error handling throughout the application is limited, making it difficult to diagnose and troubleshoot issues. Many error-handling routines simply return a Boolean value (true or false) instead of logging meaningful error messages. This practice prevents developers from identifying the underlying causes of failures and increases debugging time.

There is also a tendency for the code to follow a “happy path” mentality, meaning that it assumes operations will succeed without adequately handling failure cases. While some exceptions are caught within try/catch blocks, they often fail to return actionable error messages, further complicating debugging.

Logging is another area of concern, as many parts of the system do not log errors in a structured way. Without a central logging mechanism, tracking issues across different components becomes challenging. Implementing Laravel’s built-in logging system or integrating third-party services such as Logstash, Papertrail, or Datadog would significantly improve error traceability.

4.4 Architecture

The application follows a standard Statamic architecture built on the Laravel framework, utilizing the Model-View-Controller (MVC) pattern. Overall, the system architecture adheres to industry best practices in several areas, particularly in its use of asynchronous processing, containerization, and external payment handling. However, some architectural choices, while functional, may require further evaluation to ensure long-term scalability and maintainability.

Asynchronous Processing with Command Handlers

The system employs a command handler structure to manage certain processes asynchronously. This is primarily used for:

- Sending emails
- Processing subscriptions in Campaign Monitor
- Generating reports

By handling these operations asynchronously, the system reduces user-facing delays and improves responsiveness. Instead of processing these tasks in real time within the main

application thread, they are executed in the background, allowing users to continue their interactions without disruption. This architecture aligns with best practices for scalability and performance optimization, ensuring that resource-intensive tasks do not impact real-time interactions.

External Payment Handling via Stripe

Payments within the system are processed externally through Stripe, rather than being handled within the application itself. This approach ensures that:

- Encryption and security compliance are handled by Stripe, reducing the burden of managing PCI-DSS compliance internally.
- Payment complexity is minimized, as sensitive payment information does not need to be stored within the application.

This is a recommended best practice for handling payments, as it offloads security and compliance concerns to Stripe, a trusted third-party payment processor. However, it was not possible to verify whether the system properly synchronizes payment statuses with Stripe in real time. If payments are managed through webhooks, it is essential to ensure error handling and logging are in place to prevent discrepancies in payment statuses.

Containerization with Docker

The application utilizes Docker for containerization, which benefits both developer setup and hosting environments. Docker provides:

- Consistent development environments across different machines.
- Simplified dependency management, reducing the risk of “it works on my machine” issues.
- Easier deployment to cloud environments and scalability through container orchestration tools.

4.5 Performance

A series of performance tests were conducted to assess the responsiveness, efficiency, and adherence to SEO best practices across various pages of the site. Overall, the system demonstrates strong compliance with SEO and general best practices, achieving high scores in key performance evaluations. However, several underlying issues affect loading times, resource efficiency, caching strategies, and accessibility, which should be addressed to enhance user experience and system efficiency.

General Performance Issues

While no single page exhibits extreme performance degradation, several site-wide inefficiencies impact load times and user experience. One of the primary concerns is the loading of render-blocking resources on every page, including external services such as Stripe and Osano. These resources introduce delays even when they are not required on the current page.

For example, Stripe has been observed to take up to 2 seconds to load on a page when accessed for the first time by a user. As Stripe is only necessary on specific transactional pages (e.g. checkout), forcing users to download and initialize these resources on the homepage or other non-transactional pages results in unnecessary overhead. A more optimized approach would involve lazy loading these scripts only when required rather than including them globally.

Image Optimization & Layout Stability

Little to no image optimization appears to be in place across the site. This results in larger-than-necessary image files, increasing page load times. Implementing proper compression techniques (e.g., WebP format, lazy loading, and responsive image scaling) would improve loading speeds significantly.

Another issue identified is layout shifts, where content loads progressively and changes the visual structure of the page after it has started rendering. This is caused by elements loading dynamically without predefined dimensions, leading to a disruptive experience for users. Layout shifts negatively affect both user experience and SEO rankings, and can be mitigated by:

- Ensuring all dynamically loaded elements have explicit height and width attributes.
- Using CSS aspect ratios to prevent shifts when images load.
- Applying lazy loading where appropriate to balance performance and visual stability.

Caching Strategy for Static Content

The site employs an inefficient caching policy for static assets, including JavaScript, CSS, and font files. Instead of leveraging long-term caching to reduce redundant downloads, these files are frequently reloaded, increasing bandwidth usage and slowing down repeat visits.

To optimize performance, the system should implement better cache-control policies, ensuring that static assets are:

- Cached for longer durations in the user's browser.
- Served via a Content Delivery Network (CDN) to improve load times and reduce server requests.
- Versioned properly to ensure that updates are reflected without forcing unnecessary downloads of unchanged files.

Accessibility & Usability Considerations

The system demonstrates good overall accessibility, but several issues have been identified that may impact usability for individuals relying on assistive technologies. While a full accessibility audit was not conducted, preliminary testing revealed areas where improvements are needed.

One of the most notable concerns is form elements lacking associated labels. This omission can make it difficult for users relying on screen readers to understand the purpose of input fields. Additionally, some heading elements are not used in a sequential order, which can disrupt navigation for screen readers and impact page structure clarity.

Another significant accessibility issue relates to keyboard navigation, particularly in the site's navigation bar. When navigating using the Tab key, the current behaviour does not:

- Follow a logical sequence when interacting with dropdown menus. Instead of opening the dropdown and moving through its visible elements, the focus skips through all hidden items.
- Provide clear visual indicators when a user navigates to the navbar. This means a keyboard user may not realize they are interacting with the menu, leading to confusion and difficulty in site navigation.

4.6 Testing

Testing within the application is extremely limited, with minimal test coverage focusing only on specific services. Key components such as controllers, models, and command handlers are entirely untested, and no feature, integration, or API tests are in place. This lack of comprehensive testing introduces significant risks in terms of maintainability, stability, and the ability to detect regressions before deployment.

Current State of Testing

The existing test coverage is confined to a small subset of service-level tests, primarily covering:

- Duplicate email prevention
- Setting areas of interest
- Organization creation and relationships

While these tests provide some validation for specific business logic, they are not sufficient to ensure the stability of the overall application. Additionally, test data is hardcoded, which results in brittle tests that may fail when underlying data changes. The absence of dynamically generated test data increases the likelihood of missing test cases, reducing the effectiveness of the current test suite.

Missing Testing Layers

The following key areas lack automated testing:

- Feature Tests – There are no tests verifying that core user flows (e.g., logging in, completing a purchase, or submitting a form) function as expected. This makes it difficult to catch regressions in business-critical features.
- Integration Tests – No tests exist to validate interactions between different components of the system, such as database queries, external API calls (e.g., Stripe or Campaign Monitor), or authentication mechanisms. This increases the risk of undetected failures when integrating with third-party services.
- API Tests – Since the application includes API interactions, the absence of API tests means there is no validation of request/response handling, error handling, or authentication mechanisms. This is a major concern, particularly for ensuring API stability and correctness.
- Controller Tests – No tests are written to ensure that controllers handle requests correctly, validate input properly, and return expected responses. This means that even simple changes to a controller could introduce undetected bugs.

- Model Tests – Core business logic within models is not directly tested, meaning errors in database interactions, relationships, or computed properties may go unnoticed.
- Command Tests – Background jobs and command handlers are not tested, increasing the risk that tasks such as email sending, reporting, or data imports may fail silently in production.

Implications of Limited Testing

The lack of comprehensive testing introduces significant risks, including:

- Higher likelihood of regressions: Without automated tests, even minor code changes could break existing functionality without being detected.
- Increased debugging time: Developers must rely on manual testing, which is time-consuming and prone to human error.
- Reduced confidence in deployments: The absence of a structured test suite means new releases carry a higher risk of introducing undetected bugs.
- Greater difficulty refactoring code: Without tests to validate expected behaviour, refactoring becomes riskier and harder to execute safely.

4.7 Dependencies

A review of the system's dependencies identified several critical issues related to outdated and unsupported software versions. Many core technologies in use have reached end of life (EOL), meaning they no longer receive security updates, performance improvements, or official support. This introduces significant security, compatibility, and maintainability risks for the application.

End-of-Life Core Technologies

Statamic v2 (EOL – August 2023)

The system is built on Statamic v2, which officially reached end of life in August 2023. As a result:

- It no longer receives security patches or updates, making it increasingly vulnerable to exploits.
- Newer versions of Laravel and PHP are not fully compatible with Statamic v2, meaning any attempt to upgrade Laravel would likely require significant rework.
- Third-party add-ons and plugins for Statamic v2 are no longer actively maintained, limiting future extensibility.

PHP 7.1 (EOL – December 2019)

The system runs on PHP 7.1, which reached end of life in December 2019. This poses severe security risks, as it:

- No longer receives security updates, leaving the application exposed to vulnerabilities.
- Is not supported by many modern frameworks and libraries, increasing the difficulty of integrating new features.
- May become incompatible with future hosting environments, requiring urgent upgrades to maintain platform stability.

MySQL 5.7 (EOL – October 2023)

The database system is powered by MySQL 5.7, which reached end of life in October 2023. The primary risks associated with this include:

- No further security patches or performance optimizations.
- Potential incompatibility with future server environments, requiring migration to MySQL 8.0 or an alternative database system.
- Lack of support for newer database features, which may limit performance and functionality improvements.

These three end-of-life technologies alone present a critical risk to system security, maintainability, and long-term viability.

Front-End Dependencies – Security Vulnerabilities

An audit of the front-end dependencies revealed 321 vulnerabilities across 993 packages, with 55 Critical vulnerabilities and 172 High-severity vulnerabilities.

These vulnerabilities may include cross-site scripting (XSS), prototype pollution, insecure dependencies, and outdated JavaScript libraries. Left unaddressed, these security flaws could expose user data, compromise system integrity, and introduce exploitable attack vectors.

CMS Dependencies – Security & Abandoned Packages

Within the CMS, a total of 44 security vulnerabilities were identified across 16 packages. In addition to these security risks, 11 of these packages have been abandoned by their original maintainers, meaning they:

- No longer receive updates or security patches.
- Have no direct replacement or upgrade path, making migration more complex.
- Could introduce compatibility issues with newer versions of PHP, Laravel, or Statamic.

Implications of Outdated Dependencies

The continued reliance on end-of-life software and vulnerable dependencies introduces significant security and operational risks, including:

- Increased Exposure to Security Threats: Without security patches, known vulnerabilities remain exploitable, making the system more susceptible to data breaches, malware injections, and unauthorized access.
- Limited Support & Maintenance Challenges: With no vendor support, resolving issues and implementing improvements becomes more difficult and time-consuming. Finding developers experienced in maintaining outdated technologies may also become increasingly challenging.
- Compatibility Issues with Modern Technologies: Upgrading or integrating modern libraries, third-party services, and APIs may not be feasible due to version

incompatibilities. Newer hosting environments may drop support for older PHP and MySQL versions, forcing migration under urgent conditions.

- Regulatory & Compliance Risks: Many industry security standards (e.g., GDPR, PCI-DSS) recommend or mandate the use of actively maintained software. The presence of known security vulnerabilities in outdated packages may put the organization at legal or compliance risk if a data breach occurs.

4.8 Database

The database implementation follows standard Eloquent ORM (Object-Relational Mapping) practices and is generally structured and functional, however several inefficiencies and potential security concerns have been identified. These include string-based queries, inefficient query patterns, performance issues due to N+1 queries, and inconsistencies in indexing strategies. Addressing these areas would likely improve database performance, maintainability, and security.

Use of String-Based Queries

A number of database interactions rely on string-based raw SQL queries rather than leveraging Eloquent's built-in query builder. While this approach can sometimes provide greater flexibility, it introduces several concerns:

- Maintainability Issues: Manually written SQL queries are harder to manage, especially as the system evolves. Unlike Eloquent queries, string-based queries do not benefit from Laravel's automatic query optimizations and schema abstraction layers.
- Security Risks: If string-based queries are not properly parameterized and escaped, they increase the risk of SQL injection attacks. While Laravel provides built-in query binding to prevent injection, improper usage of raw SQL can bypass these protections.
- Inconsistent Query Structure: Some queries use Laravel's query builder, while others rely on raw SQL, creating inconsistencies in implementation across different areas of the application.

N+1 Query Issues and Relationship Handling

Several instances of nested loops accessing relationships have been observed, which can lead to N+1 query problems. This occurs when each iteration of a loop makes a new database query instead of retrieving all related records in a single optimized query.

For example, if a query retrieves a list of users and then separately fetches each user's associated records (e.g., orders, memberships, or notifications) in a loop, it can result in dozens or even hundreds of unnecessary queries. This significantly impacts database performance, especially as data volume increases.

Use of Chunking for Large Data Queries

Chunking has been implemented in some but not all queries, particularly in areas such as notifications and data exports. Chunking is a performance optimization technique that processes large datasets in smaller batches, reducing memory usage and preventing timeouts.

However, not all large queries are utilizing chunking, meaning that some operations could be retrieving large datasets into memory at once, leading to performance degradation and potential memory exhaustion.

4.9 Documentation

The documentation provided for the system is clear and well-structured, offering a high-level overview of the application and its architecture. Key system components, hosting details, and core user flows have been documented, aiding in the onboarding process for new developers. However, there are significant gaps in code-level documentation and API documentation, which could impact maintainability, debugging efficiency, and third-party integration efforts.

Provided Documentation

The documentation package includes several key resources that offer a structured overview of the system's architecture and functionality:
Entity Relationship Diagram (ERD) – Provides a visual representation of the database schema, illustrating table structures and relationships.
Architecture Diagram – Details the system's overall design, hosting environment, and key integrations, helping developers understand how different components interact.
Core User Flows – Outlines the primary user journeys within the system, offering insight into how users navigate and interact with features.
Hosting & Infrastructure Documentation – Provides technical details on the hosting environment, including server configurations, deployment processes, and dependencies.
Developer Onboarding Guide – Offers step-by-step instructions on setting up the development environment and getting the system running.

These documents provide essential context for developers working on the system and are valuable in ensuring that new team members can quickly onboard and understand the system's architecture.

Lack of Code-Level Documentation

While the high-level documentation is sufficient for understanding the overall system, the codebase itself lacks inline documentation and explanatory comments. Inline comments appear sporadically, and there is little explanation of complex business logic, making it difficult for developers to understand the purpose of key functions and classes, identify dependencies between components and quickly debug or modify existing logic without extensive manual tracing.

Well-documented code is critical for ensuring long-term maintainability, especially in a system with custom implementations and legacy components. Without proper documentation, developers may need to spend significant time reverse-engineering functionality, increasing the risk of unintended changes or errors.

4.10 Security

A preliminary security review of the system found no immediately exploitable vulnerabilities in terms of unauthorized access or authentication bypasses. The application relies on Laravel's and Statamic's built-in authorization mechanisms, which generally provide robust access control. However, the security audit revealed several high and medium-risk vulnerabilities that require attention.

Authorization & Access Control

The system relies heavily on Laravel and Statamic's built-in security features, particularly for authorization and access control. While key endpoints appear to be restricted appropriately, there is minimal additional server-side validation beyond what the framework provides. This means that access control is dependent on Laravel's default mechanisms, with no additional backend verification of authorization levels before executing critical actions. Administrator endpoints should enforce additional backend validation, even if frontend role-based access control (RBAC) is in place.

Although this is not an immediate security risk, best practices dictate that authorization checks should always be enforced at the backend, particularly for sensitive operations.

Additionally, static assets within the Main Assets directory are publicly accessible without authentication, meaning any file within that directory can be accessed directly via URL, regardless of user privileges. This could lead to unintended data exposure if sensitive files are mistakenly placed within this directory.

Security Audit & Identified Issues

A full security audit has not yet been conducted, but basic penetration testing and vulnerability scans were performed to identify potential weaknesses. The following security concerns have been detected, categorized by severity. Issues identified as Low or below are not included.

Name	Risk Level	Number of instances	Description	Impact
Cross-Site Scripting (XSS) - Reflected	High	1	The system echoes attacker-supplied code into the browser without proper sanitization, allowing arbitrary JavaScript execution within a user's session.	If exploited, an attacker could steal session cookies, redirect users to malicious sites, or modify page content.
SQL Injection	High	12	A vulnerability was detected where SQL queries are being executed with unvalidated input, allowing a modified query to execute on the database.	If exploited, attackers could manipulate database queries, exfiltrate sensitive data, or delete records.
Content Security Policy (CSP) Header Not Set	Medium	1306	The system lacks a Content Security Policy (CSP) header, which is used to restrict which sources can execute scripts, load styles, and embed media.	This increases the risk of Cross-Site Scripting (XSS), Clickjacking, and Data Injection attacks.
Missing Anti-clickjacking Header	Medium	1108	The response does not protect against 'ClickJacking' attacks. It should include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options.	This allows attackers to embed the site in an iframe, potentially tricking users into unintended actions such as clicking buttons, submitting forms, or exposing sensitive information.

4.11 Administration Panel – Risk of Unrestricted Access

The administration panel allows all administrators to modify critical system settings, including filesystem structure, license keys, and global formatting settings (Redactor settings). Allowing unrestricted access to these settings increases the risk of accidental misconfigurations, which could cause site-wide failures.

To prevent this, tiered admin roles should be introduced, ensuring that only super administrators can modify critical settings, while content managers have restricted access.

4.12 Stripe Integration & Pricing Risks

There are potential risks in how pricing and fees are managed within the system. Although access to the Stripe instance was not available for review, it appears that product pricing and fees are controlled within the CMS rather than within Stripe's own dashboard.

If pricing is hardcoded in the database, there is a risk that manual errors or outdated pricing could affect live transactions. A more robust solution would be to integrate pricing directly with Stripe and manage costs dynamically through its API to avoid discrepancies. This would also allow the future use of Stripe's built in pages for users to manage their payment information and retrieve invoices for previous purchases.

5. Redevelopment Strategy

It is recommended that the application undergo a full rebuild rather than attempting to refactor or patch the existing codebase. The current system presents significant technical debt, security vulnerabilities, and long-term maintainability challenges due to outdated dependencies, inefficient architecture, and a lack of flexibility in key areas. A rebuild will allow the organization to modernize its infrastructure, improve system performance, and streamline administrative workflows while maintaining all essential functionality.

The following considerations outline the key factors in the rebuild strategy, including data migration, feature prioritization, payments handling, security improvements, and hosting considerations.

5.1 Rebuild vs Refactoring

While refactoring could address some issues, the system's reliance on outdated and unsupported technologies introduces inherent risks. Additionally, critical security vulnerabilities—including SQL injection risks, lack of modern authentication practices, and missing security headers—further support the case for a fresh implementation using modern frameworks and best practices.

A rebuild will:

- Eliminate technical debt by using a modern and supported technology stack that ensures long-term maintainability.
- Improve maintainability and performance by replacing inefficient queries, fixing data retrieval issues, and implementing better caching strategies.
- Enhance security by enforcing stronger authentication mechanisms, API protection, and updated dependency management.
- Streamline workflows by removing unnecessary complexity from content management and shifting financial logic into Stripe for automation.

A refactor would require extensive work to modernize the system while still dealing with legacy constraints, making a full rebuild the more cost-effective and future-proof solution.

5.2 Data Migration Strategy

A critical component of the rebuild will be ensuring that essential data is preserved and migrated efficiently. The following key questions must be addressed:

- What historical data must be retained?
    - Member accounts, purchase history, certification records, and event attendance should be migrated.
    - Older, less relevant data (e.g., expired job board listings, archived content) should be reviewed for potential exclusion.
- How will data be transformed?
    - Existing database relationships may need restructuring to align with modern best practices.

    - Data should be cleaned to remove duplicate or inconsistent records before migration.
- What approach will be used?
    - A staging environment should be set up to test data migration before launching the new system.
    - Automated scripts should be created for migrating users, memberships, events, and purchases from the old system to the new.

Proper planning will ensure a smooth transition with minimal disruption.

5.3 Feature Prioritization & Rationalization

The rebuilt system must support all critical business functions, but this presents an opportunity to streamline content management and remove unnecessary complexity.

Key Features to Retain & Improve

- Membership Management – Ensure smooth registration, renewals, and access control.
- Events & Training – Maintain the ability to list, book, and manage training and online and in-person events.
- Payments & Subscriptions – Move all pricing logic into Stripe to simplify administration.
- Reporting & Analytics – Improve the reporting system by making data exports more flexible.
- Admin Panel Navigation – Redesign admin workflows to reduce time spent on repetitive tasks.
- Authentication – Utilise a third-party Authentication provider to simplify administration and ensure best practices

Features That Could Be Removed or Simplified

- CMS Flexibility for Certain Sections – Sections such as Event Descriptions, Training Descriptions, and Award Content do not require full CMS flexibility. Instead, templated structures with input fields should be used for consistency and ease of use.
- Job Board – If underutilized, the job board could be reviewed for removal or replaced with a more streamlined posting system.

Key Improvements in the Rebuild

- Implement structured content management where full CMS flexibility is not needed to reduce maintenance overhead.
- Introduce bulk management tools for adding events, courses, and reports to reduce administrative workload.

5.4 Payments & Financial Handling

Currently, financial logic (event fees, member fees, training fees) is stored within the administration system, making it difficult to update, track, and manage payments efficiently.

To streamline this, all pricing and transaction management should be migrated to Stripe, leveraging its built-in invoicing, subscription, and checkout pages. This will:

- Ensure consistency in pricing by storing fees directly in Stripe rather than in the internal system.
- Reduce administrative overhead, as price adjustments and invoice generation can be handled directly in Stripe.
- Improve security and compliance, as Stripe fully manages PCI compliance, removing the need to store payment details internally.

The administration panel should dynamically pull pricing from Stripe, allowing adjustments without requiring code changes.

5.5 Development & Implementation Strategy

- Conduct a separate, dedicated UX review phase to ensure requirements and user journeys are fully defined before development begins.
- Employ phased, iterative delivery:
    - **Phase 1 (MVP)**: Core functionality (user journeys, essential payments, CMS templates)
    - **Phase 2**: Enhanced administrative automation and detailed reporting features
    - **Phase 3**: Additional "nice-to-have" features based on user feedback and priorities
- Establish regular milestones and clearly defined deliverables at each stage, validated by the Technical Consultant and stakeholders.

5.6 Testing & Launch

- Clearly define User Acceptance Testing (UAT) scenarios early in the project.
- Conduct rigorous testing of:
    - Functional correctness (core user journeys and administrative workflows)
    - Performance under expected peak conditions (membership renewals, event bookings)
    - Security and compliance checks (penetration testing, GDPR compliance)
- Plan and execute a structured launch approach, including communication to users, clear rollback strategies, and post-launch support measures.

5.7 Post-launch & Continuous Improvement

Schedule regular review meetings post-launch to assess performance against initial objectives, identify improvement areas, and plan further enhancements.

6. Recommendations

6.1 UX review

A dedicated UX review phase should precede the development phase. This approach ensures clarity on user journeys, optimal site navigation, and improved overall user experience. It will also help identify potential usability issues early, minimising costly revisions later in the project. Ideally, this would be separate to the development phase, so that costs can be re-adjusted based on feedback from the UX phase.

Clearly Defined User Journeys

Particular attention should be given to simplifying and enhancing the following key user journeys:

Membership Registration & Renewal

- Simplify initial registration processes to reduce drop-off rates.
- Clearly communicate the benefits of membership and renewal procedures.
- Provide clear steps and feedback throughout the renewal process.
- Introduce automatic assignment to Group Membership via associated email domain, with optional approval based on level of membership

Event & Training Booking

- Streamline the booking process to ensure minimal steps from discovery to confirmation.
- Ensure seamless integration with external webinar links and provide clear, intuitive event notifications (ICS calendar integration).
- Investigate the use of and integrating with third-party solutions to improve user interaction as well as administration

Payments & Invoicing

- Ensure transparent communication during payment processes.
- Provide an intuitive and secure payment checkout experience, fully integrated with Stripe or other payment provider.
- Clearly indicate payment statuses and provide easily accessible invoices and receipts for the customer.
- Ensure that invoice chasing and delayed payment are handled automatically, and the burden is significantly reduced on BHBIA staff.

User Interface & Accessibility Improvements

To enhance user experience and accessibility, the rebuild should address these specific issues identified:

Form Design & Validation

- Ensure all forms have clearly associated labels to improve accessibility.
- Implement robust inline validation and clear error messaging to improve form usability.

Layout Stability & Image Optimisation

- Prevent layout shifts by defining image and content dimensions explicitly.
- Optimise images (e.g., responsive images, WebP format, lazy loading) to improve load times.

Navigation & Content Discoverability

- Revisit the main navigation structure to enhance user discoverability of content such as Events, Training, and Members Directory.
- Simplify and clarify site navigation, particularly on mobile devices.

Accessibility Compliance

- Ensure compliance with accessibility standards (e.g., WCAG 2.1 guidelines).
- Address issues such as inconsistent heading structures and inadequate keyboard navigation for dropdown menus and interactive elements.

6.2 Content, CMS & Administration

Structured CMS Templates

- Implement structured, predefined content templates for frequently used content areas such as News, Events, Resources, and BOBI Awards.
- Reduce unnecessary complexity currently caused by overly flexible, unstructured content pages, making it easier for administrators to manage content consistently.
- Templates should be designed so that there are multiple options for page designs, that they are responsive and don't need to be manually adjusted by an administrator based on images or content.

Content Management Efficiency

- Adopt a clearly defined workflow for content creation, editing, and management to enhance administrative efficiency.
- Ensure that content cannot be edited at the same time, to ensure data isn't lost.
- Ensure that preview of content that isn't live yet is correct.

Administrative Workflow Improvements

- Clearly define administrative roles and permissions, establishing tiered access (e.g., Super Admin vs. Content Admin) to prevent unintended changes to critical system settings.
- Simplify admin workflows through bulk management capabilities, especially for routine tasks such as adding events or updating training modules.
- Improve historical data visibility and reporting on administrative areas such as payments, membership status, and subscription history.

Enhanced Reporting System

- Rebuild the reporting system to offer greater flexibility, eliminating the need to manually create numerous templates for slight variations.
- Ensure reports can aggregate data from multiple database tables simultaneously, improving ease of auditing and data accuracy.

- Introduce numerous pre-defined templates for regular reporting needs. These should be defined by BHBIA, including the columns required. If regularly running reports, then investigate automation of the reports.

Asset Management Improvements

- Transition to efficient asset storage and management (cloud-based or otherwise) to address scalability issues currently posed by asset management via Version Control.
- Ensure that document and image management within the CMS is intuitive, structured, and performant.
- Ensure assets are organised in the CMS and are reusable.
- Ensure that assets cannot be accessed if they are only accessible on private pages, such as paid for events and training, or are explicitly for the use by the board.

SEO and Content Discoverability

- Enhance content management tools to support robust SEO features, including metadata management, structured data markup, and optimised URLs.
- Implement proper caching strategies, leveraging a CDN for static assets to improve site performance and user experience.
- Ensure that whatever Front-end framework is chosen, search engine crawlers can access individual pages for indexing.

6.3 Technical Approach

Recommended Technical Stack

Based on the audit findings, the new system should be built on a modern, sustainable, and scalable technology stack. Recommendations include

A modern CMS platform.
This can be Headless (i.e. not utilising the CMS's own front-end) or not. Examples include WordPress, Shopify or Contentful.

Modern Front-end Framework
Utilise React or Vue.js to create a performant, scalable, and responsive front-end. This approach allows better separation of concerns between front-end content rendering and backend administrative functions.

Utilisation of off-the-shelf components
Use third-party components for Authentication and Payment integration, as well as investigating use of a third-party Learning Management System (LMS) for the creation and implementation of Online Training, as well as an Event System for Event Management. Need to ensure that they are integrated well and are of a benefit to the user, rather than just reducing development and support cost.

Centralised Administration

- Establish a clear separation between administrative functions and the user-facing website by using a dedicated admin subdomain (e.g., admin.bhbia.org.uk).
- Ensure admin access is tiered and secure, following the principle of least privilege.

Security Enhancements

- Implement robust authentication and authorisation measures (e.g., Multi-Factor Authentication, role-based access control).
- Require strict input validation and secure handling of API endpoints.
- Ensure compliance with modern web security standards (CSP headers, protection against XSS, and SQL injection prevention).

Hosting & Infrastructure Considerations

- Infrastructure should support auto-scaling during high-demand periods, ensuring optimal site performance, whilst balancing cost implications.
- Include a clear, automated CI/CD pipeline with rollback capabilities.
- Ensure client retains ownership and direct administrative control over hosting to provide flexibility in supplier management.

Reporting

- Reporting by an administrator should be flexible enough that they can retrieve the data they need, without being too complicated that development becomes an issue.
- First rely on templates, and increase flexibility based on requirements
- Ensure that historical data is retained and that reporting on the historical audit trail of changes in membership subscriptions, payments and other specific data is possible.

Data Migration Strategy

A careful migration strategy will be critical. The approach should include:

- Identification and prioritisation of essential data
    - User profiles and membership history
    - Event attendance and training completion history
    - Historical financial transaction records
- Pre-migration data cleansing to remove inconsistencies and duplicates, and archived data that doesn't need to be retained.
- Use of a staging environment to perform dry-run migrations for validation.
- Development of automated scripts to facilitate a reliable migration process.

6.4 Payment Integration & Financial Handling

- Centralise all payment processes within Stripe or other payment provider, ensuring a single, unified platform for payment management.
- Utilise Stripe's built-in invoicing and subscription management features.
- Automate debtor management, including automated reminders for overdue payments.
- Integrate payment exports directly with accounting software (Sage) to reduce manual reconciliation.
- Ensure clear, user-friendly access for users to their invoices and payment history.

6.5 User Management

- Simplify the member registration process to reduce user friction and abandonment.

- Improve management capabilities for organisation administrators, allowing easy management and reporting of members.
- Facilitate streamlined user movement between companies, maintaining historical records clearly and accurately.
- Enhance visibility of membership history, including event attendance, training completion, and financial transactions.
- Clearly define user access rights and administrative roles, simplifying overall user administration.

6.6 Security & Compliance

- Implement OWASP Top 10 Security mechanisms, ensuring a secure application.
- Consider advanced authentication mechanisms such as Multi-Factor Authentication (MFA) for administrative users.
- Review integrations, especially with email marketing (e.g., Campaign Monitor), to ensure GDPR compliance.
- The website should be subject to a Penetration Test before going live, which should be included in the cost quoted by the supplier.

6.7 Campaign and Marketing

- Identify a way of segregating users and what marketing and campaign's need to be sent. These should be set up in Campaign Monitor or other Campaign email system.
- Ensure that you have complete access to the platform rather than having to utilise it through the administration panel
- Be careful about how flexible this is. Allow users to select their own preferences, such as Areas of Interest, Company types, Event types etc. Allow a user to consent to what type of marketing they would like and allow for them to unsubscribe from one type without affecting others.

6.8 Deployment and Go Live

- Ensure that deadlines are not structured around recertification timelines, and that Go Live should be aimed for a short while after recertification has happened, to reduce issues.
- A Go Live plan should be constructed to ensure a smooth transition, and a hyper care support agreement in place for the first week of initial Go Live.
- Design an iterative release strategy so that the minimal viable product (MVP) is released first (which may include a regression in features initially), with continually releases after that. Try and avoid a full overhaul with many moving parts, however this may be unavoidable.

7. Risks & Mitigations

7.1 Data Migration Complexity and Integrity Issues

Description
Migrating legacy data (membership details, payment history, training records, and event data) may lead to data integrity issues or loss.

Impact
Data inaccuracies or loss could negatively impact user trust, financial reporting, and historical record management.

Mitigation
- Prioritise and cleanse data thoroughly before migration.
- Employ scripted migrations with validation processes in staging environments.
- Conduct comprehensive User Acceptance Testing (UAT) to validate data integrity.

7.2 Inadequate Security Measures

Description
Previous system vulnerabilities (SQL injection, XSS, outdated components) may persist without stringent oversight and clear security standards.

Impact
Compromised user data, reputational damage, regulatory non-compliance, and potential financial penalties.

Mitigation
- Enforce secure coding standards and regular code reviews.
- Require comprehensive security testing from suppliers, including penetration testing.
- Clearly outline security and compliance requirements.

7.3 Overly Flexible Requirements

Description
Excessive flexibility in content and administrative areas has previously resulted in complexity, high maintenance costs, and administrative inefficiency.

Impact
Continued complexity, increased maintenance overhead, difficulty managing content consistently.

Mitigation
- Adopt structured CMS templates for repetitive content.
- Clearly specify areas that require flexibility versus those that can remain templated or standardised.
- Maintain strict governance around content structure decisions.

7.4 Payment Integration and Financial Automation Challenges

Description
Fragmented financial management, manual debtor management, and limited integration previously led to inefficiencies.

Impact
Continued manual processes, increased administrative overhead, and potential financial inaccuracies.

Mitigation
- Clearly specify integration requirements with Stripe and accounting software (Sage).
- Implement robust automation for debtor management, invoicing, and financial reporting.
- Validate integrations extensively before launch.

7.5 Performance Issues During Peak Periods

Description
Current website faces severe performance degradation during membership renewal and event booking peaks.

Impact
Poor user experience, lost revenue opportunities, and reduced trust in system reliability.

Mitigation
- Clearly define performance benchmarks and expectations.
- Select scalable hosting infrastructure with built-in performance monitoring and autoscaling capabilities.
- Conduct thorough load testing and performance tuning prior to deployment.
- Ensure that the admin site is separate to the Main site

8. Appendix

8.1 Security Audit report

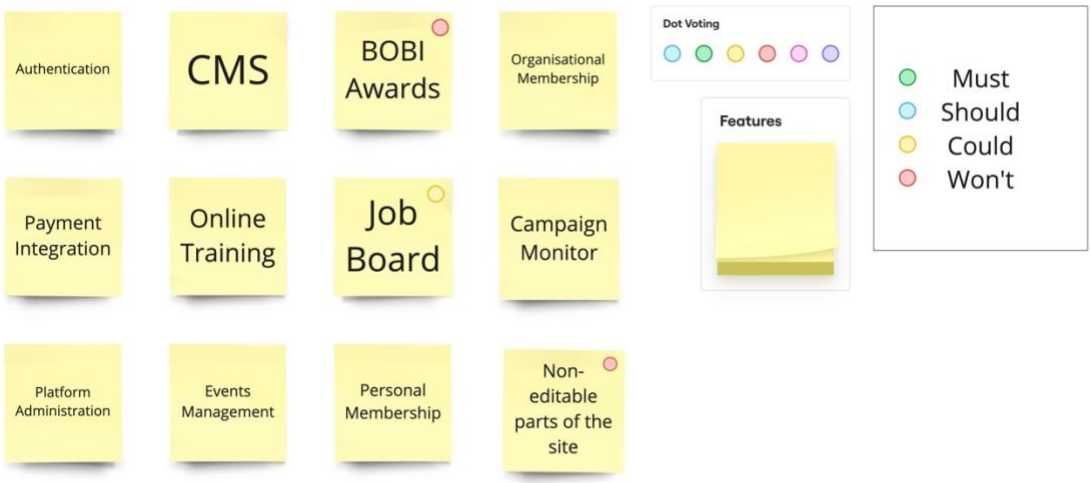
Vulnerability Impact

#	Name	Impact
1	Cross Site Scripting (Reflected) [1] [2]	Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.
2	SQL Injection - SQLite [1]	SQL injection may be possible.
3	Content Security Policy (CSP) Header Not Set [1] [2] [3] [4] [5] [6] [7]	Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.
4	Cross-Domain Misconfiguration [1]	Web browser data loading may be possible, due to a Cross Origin Resource Sharing (CORS) misconfiguration on the web server.
5	Missing Anti-clickjacking Header [1]	The response does not protect against 'ClickJacking' attacks. It should include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options.
6	Vulnerable JS Library [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]	The identified library jquery, version 1.11.3 is vulnerable.
7	CSP: Notices [1] [2] [3] [4] [5]	Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks. Including (but not limited to) Cross Site Scripting (XSS), and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.
8	Cookie No HttpOnly Flag [1]	A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible.
9	Cookie with SameSite Attribute None [1]	A cookie has been set with its SameSite attribute set to "none", which means that the cookie can be sent as a result of a 'cross-site' request. The SameSite attribute is an effective counter measure to cross-site request forgery, cross-site script inclusion, and timing attacks.
10	Cookie without SameSite Attribute [1]	A cookie has been set without the SameSite attribute, which means that the cookie can be sent as a result of a 'cross-site' request. The SameSite attribute is an effective counter measure to cross-site request forgery, cross-site script inclusion, and timing attacks.
11	Cross-Domain JavaScript Source File Inclusion	The page includes one or more script files from a third-party domain.
12	Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) [1] [2]	The web/application server is leaking information via one or more "X-Powered-By" HTTP response headers. Access to such information may facilitate attackers identifying other frameworks/components your web application is reliant upon and the vulnerabilities such components may be subject to.
13	Server Leaks Version Information via "Server" HTTP Response Header Field [1] [2] [3]	The web/application server is leaking version information via the "Server" HTTP response header. Access to such information may facilitate attackers identifying other vulnerabilities your web/application server is subject to.
14	Strict-Transport-Security Header Not Set [1] [2] [3] [4] [5]	HTTP Strict Transport Security (HSTS) is a web security policy mechanism whereby a web server declares that complying user agents (such as a web browser) are to interact with it using only secure HTTPS connections (i.e. HTTP layered over TLS/SSL). HSTS is an IETF standards track protocol and is specified in RFC 6797.
15	Timestamp Disclosure - Unix [1]	A timestamp was disclosed by the application/web server. - Unix
16	X-Content-Type-Options Header Missing [1] [2]	The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.

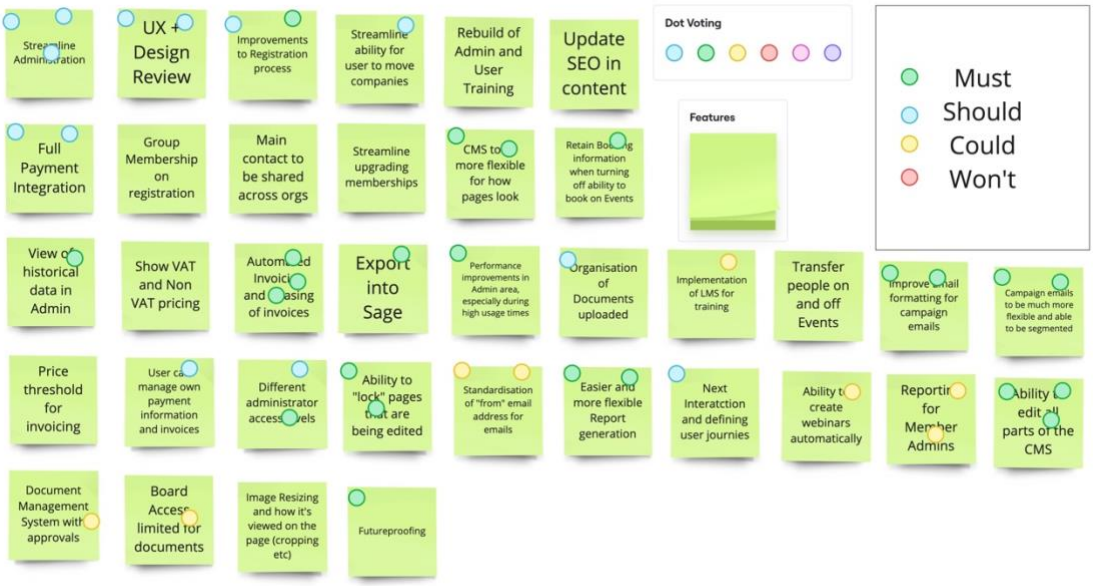
Vulnerability Descriptions

8.2 Workshop Whiteboard Outputs

Current Features



New Features



User Stories

Why do they join?

Persona 1

Background: Wants to access news and information

Tone of voice:

Name

Needs: Understand what BHBIA can offer them; How to access; Free resources, legal and ethical guidelines

Age

Desires: What is BHBIA? Browse availabe courses; What do you get if you do become a member?

Location

Fears: Don't know what they don't know; Not being compliant; Fear of not being involved

Persona 2

Background: Registered to look at Online Training

Tone of voice:

Name

Needs: Don't know their organisation is a member; Registering by mistake; Information around Guidelines

Age

Desires: What do you get if you do become a member?

Location

Fears: Bombarded by emails and information i'm not interested in; Complexity of engagement and price

Persona 3

Background: Has to do Online Training for their company

Tone of voice:

Name

Needs: Quickly and easily access and complete online training; Ability to pay for training

Age

Desires: Access training that will help them do their job and progress in their career; Gaining competitive advantage; Want to make bookings for events

Location

Fears: Can I pass the online training? How long would it take to do the training?

Persona 4

Background: Administrator of BHBIA Site

Tone of voice:

Name

Needs: Flexible, Functional and Fast Simple

Age

Desires: Logical User Journeys

Location

Fears: Too Technical; Will struggle to complete task; That it could break and will affect all users

Persona 5

Background: Organisation Owner and License Payer

Tone of voice:

Name

Needs: Easy to use; Having oversight over your organisation; Want to see invoices and payment information

Age

Desires: To look at a list of all my members, see what they're booked on and what they've done so far

Location

Fears: