



National College of Ireland

Technical Report

Title: BiggerPhish: An Interactive Cybersecurity Training Platform

Due: 04/8/2024

Programme Name: BSc in Computing

Specialisation: Cybersecurity

Academic Year: 2023/2024

Student Name: Padraig McCauley

Student Number: 20123744

Student Email: x20123744@student.ncirl.ie

Table of Contents

1. Executive Summary.....	2
2. Introduction	2
2.1 Background.....	2
2.2 Aims.....	3
2.3 Technology.....	3
2.4 Structure	4
3. System.....	5
3.1 Requirements	5
3.1.1 Functional Requirements	5
3.1.2 Data Requirements.....	12
3.1.3 User Requirements.....	12
3.1.4 Environmental Requirements	13
3.1.5 Usability Requirements	13
3.2 Design & Architecture	13
3.3 Implementation.....	19
3.4 Graphical User Interface (GUI)	29
3.5 Testing.....	37
Unit Testing	37
Security Testing	38
Functional Testing.....	39
3.6 Evaluation	41
4. Conclusions.....	42
5. Further Development or Research	42
6. References	43
7. Appendices	44
Project Proposal	44
Reflective Journals	49

1. Executive Summary

This report details the BiggerPhish educational platform's conception, implementation and points of note in relation to the techniques and technologies used to create the platform. The report lists the eight major functional requirements gathered through research and how these requirements are significant to the platform's implementation – ensuring it works as expected and also serves its purpose as an education platform effectively. Security was considered through every stage of the implementation and is detailed in both the functional requirements (example encryption specifications) as well as the testing of the platform itself – the platform must be an example of secure programming. The report describes the key components of the application with image examples as well as descriptions of the code when needed to better illustrate the flow of the system. The report describes the testing processes for each of the categories of testing types needed (unit, system, security etc.) and evaluates the system based on these results. The conclusion gives insight into both the personal and technical accomplishments of the project and later the report elaborates on where the project could potentially go if continued outside of the time constraints set. The appendix gives some insight into the project's progression through the original project proposal and the monthly reflective journals. To reiterate the conclusion of the project, the implementation was a success, based on personal metrics yet with the knowledge gained throughout the course of the project it is felt that a more robust solution could be built if reimplemented. The experience was worthwhile, and much was learned about the technologies, the topics and management of times and projects as a whole.

The platform is available at <https://www.padraigmccauley.online:3000/> and note it has a self-signed certificate so the warning must be bypassed.

Login details:

adminUser@test.com

Boat-Throw-Wing-Robbery-5

standardUser@user.com

Lazy-Beg-Report-Cruel-4

2. Introduction

2.1 Background

The initial idea for the platform was to offer some form of education for cybersecurity in a way that is novel but also technically challenging enough to satisfy my personal goals and the brief of the project. Inspiration was taken from WebGoat by OAWSP as this offers modular lessons that users interact with to complete. Having a way to 'show'

users a real-life phishing attack seems like an achievable goal and a way to offer a gamified method of education that is accessible.

The project idea was also inspired by the PC game “Papers, Please” which has users make decisions on whether a person is allowed to pass a security checkpoint based on a review of their documentation (passport, letters etc). The game has a time element which means there is a limited window to assess all materials before making a decision, so this was an interesting mechanic I wanted to incorporate as it would match a real-world scenario in the event of a phishing attack.

By mixing the practical approach of cybersecurity training from WebGoat with the time-based decision-making aspects of "Papers, Please", the platform should give a unique and novel way for users to learn about phishing attacks in a lasting way. In addition to the above I have also set a personal challenge to attempt to incorporate two database technologies – SQL and NoSQL. I am doing this to challenge my technical abilities as well as gain knowledge in an area I have wanted to improve for some time.

2.2 Aims

The key aim of BiggerPhish is to create an engaging educational platform, inspired by applications like WebGoat by OWASP and the game "Papers, Please". The project aims to achieve the following goals:

- To offer cybersecurity education in an interactive way.
- To gamify learning in order to appeal to a wide range of students.
- To track progress of content completed.
- To ensure a smooth user experience.
- To implement a platform with full administrative capabilities for platform management.
- To achieve personal goal of implementing two database types.

2.3 Technology

In developing BiggerPhish, my approach began with a comprehensive requirement identification phase. I researched similar products/materials (WebGoat, for example) to capture all the necessary functional and non-functional requirements.

Once I had enough information to determine my requirements, I adopted an agile methodology, breaking down the project into two-week sprints. This iterative approach ensured the project adapted to changes in a timely manner.

I chose Express.js (node.js) for the backend to leverage its efficiency and because it was a technology I had always wanted to improve on for professional reasons. The frontend uses HTML5, CSS3, and JavaScript to provide the interactive user experience. I also incorporated Bootstrap for its responsive design capabilities and ease of use for building the UI grids etc.

Microsoft SQL server and Mongo DB were the two database technologies used to hold data persistently and support the functionality's storage needs.

I also used GitHub for version controlling. Security is paramount in cybersecurity application, so express middleware such as Helmet and csrf were incorporated to adhere to secure coding standards.

For deployment, I set up a CI/CD pipeline using Circle CI (coupled with GitHub), which enabled me to maintain a consistent delivery of features while ensuring the application is stable and secure.

2.4 Structure

The structure of this report is organised as follows:

Introduction

This section provides a background to the project, giving context to decisions made throughout the project and detailing the technical decisions made to accomplish the objectives.

System

The system section details the functional requirements of the platform, examining each of the use cases used as key components of the application.

It explains the architecture of the solutions for these use cases, how they are implemented and how testing and evaluation has been carried out to confirm their completeness.

Conclusion

The conclusion summarises and evaluates the project, considering the successes and failures of the implementation. The further development or research section discusses the potential road map for further development of the project and suggests improvements if more time and resources are available.

Appendices

This section includes the original project proposal and reflective journals which were written throughout the course of the project, broken into months

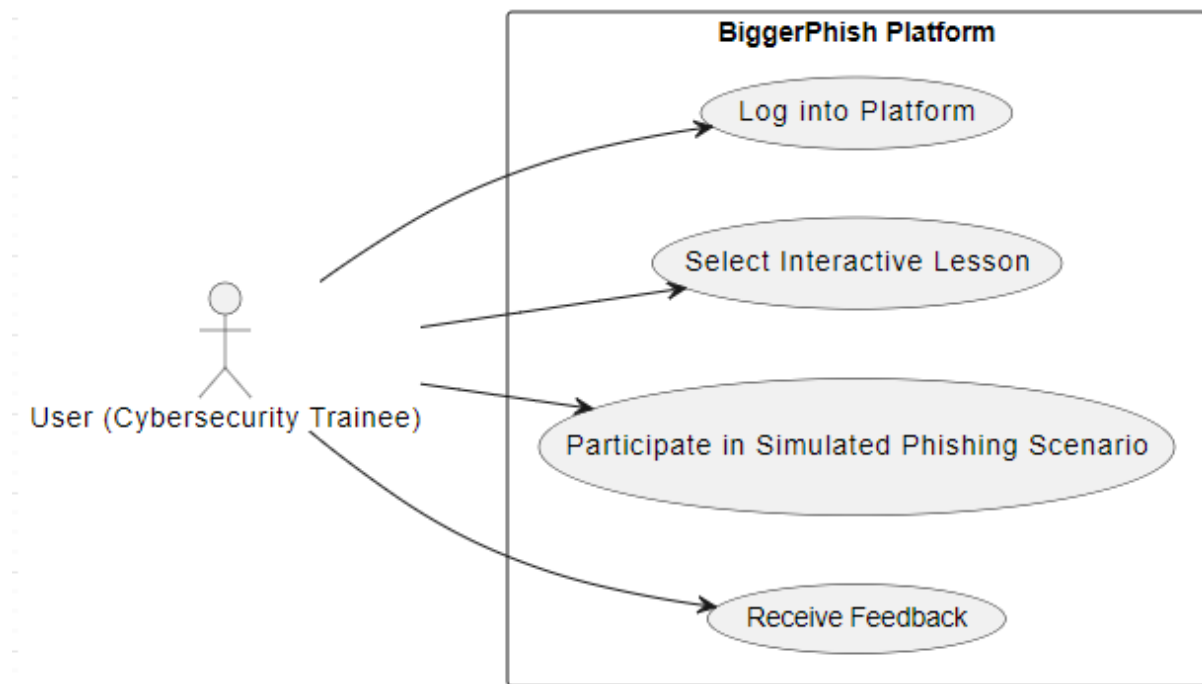
3. System

3.1 Requirements

3.1.1 Functional Requirements

3.1.1.1 Interactive Cybersecurity Education

Use Case Diagram



Description & Priority

The platform will provide users with practical, interactive lessons allowing learners to participate in simulated phishing attacks.

Priority: High

Use Case

Scope: The scope of this use case includes interactive cybersecurity lessons on phishing attacks.

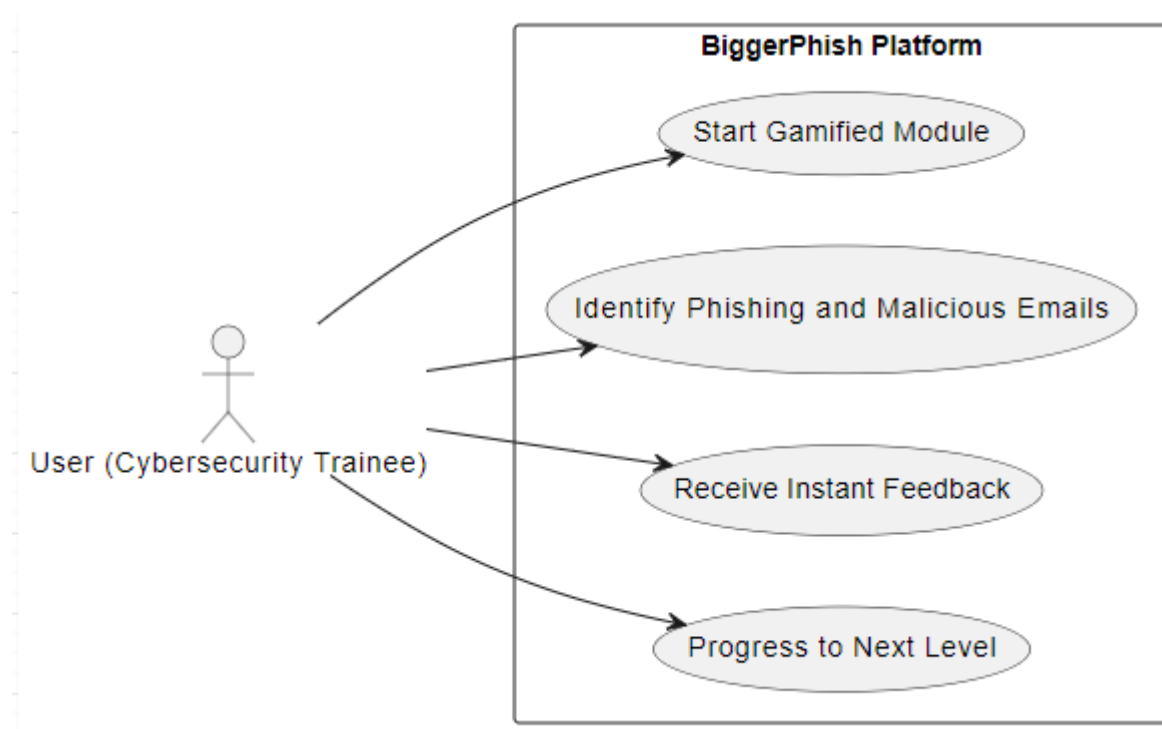
Flow Description:

- **Precondition:** Users log in.
- **Activation:** This use case starts when a user selects an interactive lesson on phishing attacks.
- **Main Flow:**
 - The system presents the user with a simulated phishing attack scenario.

- The user analyses the scenario and decides whether the email is real or fake.
- The system provides feedback on the user's decision after five emails have been flagged.
- The user proceeds to the next lesson or repeats the scenario for better understanding.

3.1.1.2 Gamification of Learning

Use Case Diagram



Description & Priority

Include game-like elements to the module activities such as time-based decision-making and a countdown timer. This involves users identifying phishing and malicious emails under time constraints, simulating real-world pressure

Priority: High

Use Case

Scope: Gamification aspects within training modules.

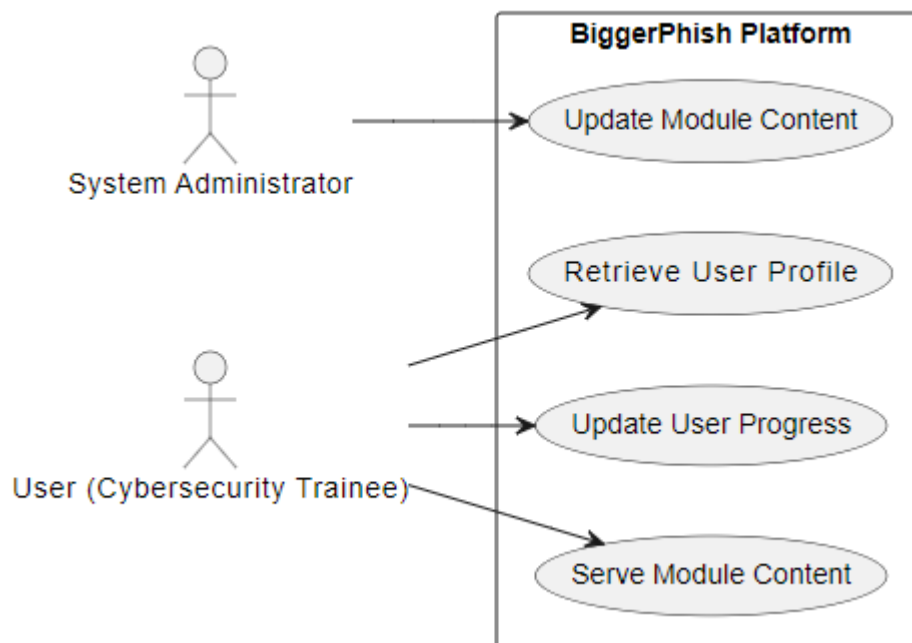
Flow Description

- **Precondition:** User has begun a gamified module.
- **Activation:** This use case starts when the module presents the user with a series of emails.
- **Main Flow:**

- The system offers a mixed set of phishing and legitimate emails.
- The user must quickly identify and sort the emails.
- The system assesses the user's performance and provides feedback in the form of a score out of five.
- The difficulty level increases as the user progresses, in the form of more subtle phishing attempts.

3.1.1.3 Dual Database System

Use Case Diagram



Description & Priority

Implement both a SQL and NoSQL database within the system. The SQL will store user and course information while the NoSQL will contain email content for the modules.

Priority: Medium

Use Case

Scope: Handling of data through a dual database system.

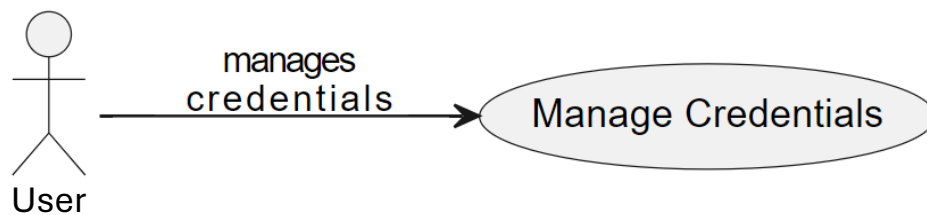
Flow Description

- **Precondition:** The system is operational with initialised databases.
- **Activation:** This use case starts when a user interacts with the platform, requiring data retrieval or storage.
- **Main Flow:**
 - The system retrieves user profiles and progress from the SQL database.

- Dynamic content for the modules is served from the NoSQL database.
- User interactions, such as completing modules or updating profiles, trigger data updates across both databases.
- The system ensures data consistency and integrity between the databases.

3.1.1.4 Credential Management System

Use Case Diagram



Description & Priority

System credentials used by this system to access databases, SMTP Server, SSH access and more stored securely.

Priority: High

Use Case

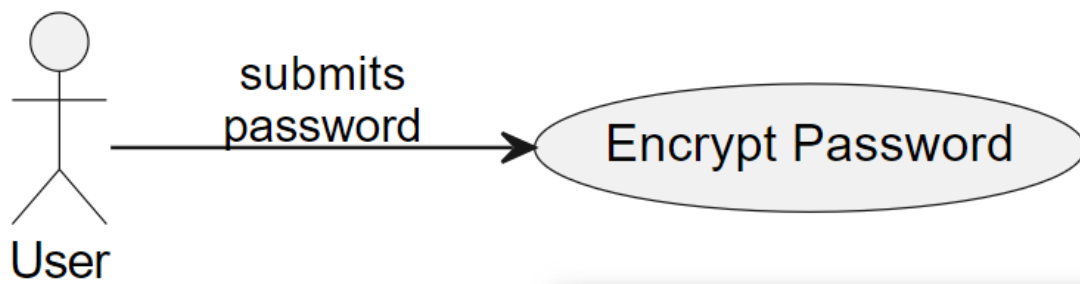
Scope: Security of system access credentials.

Flow Description

- **Precondition:** Credentials stored as environmental variables.
- **Activation:** The system accesses the environmental variables if/when access is needed to any of the components.
- **Main Flow:**
 - Credentials are stored as environment variables, not hardcoded.
 - The application accesses these credentials securely when connecting to any of the mentioned components.

3.1.1.5 Password Encryption

Use Case Diagram



Description & Priority

Uses bcrypt to encrypt passwords when stored in the database. In doing so, this prevents unauthorised access in the event of a data breach as passwords are not stored in plaintext.

Priority: High

Use Case

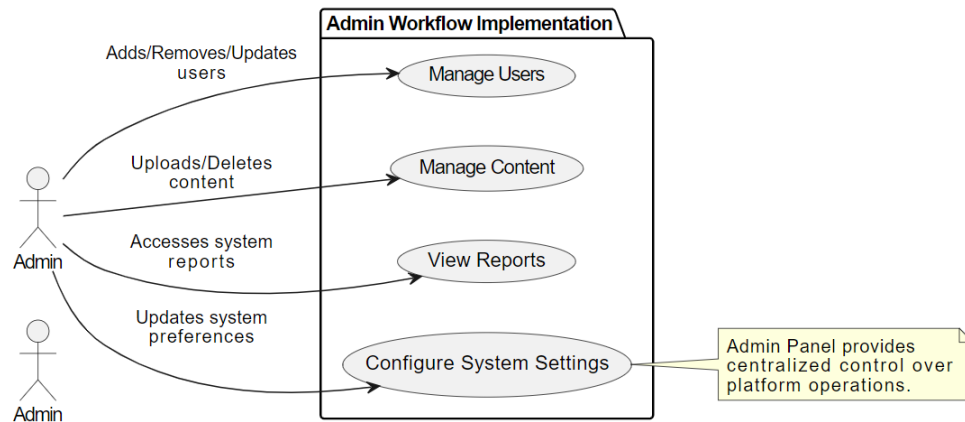
Scope: Security of user authentication data.

Flow Description

- **Precondition:** A user registers or changes their password.
- **Activation:** The user submits a password through the registration or password change form.
- **Main Flow:**
 - The system captures the plaintext password input by the user.
 - Bcrypt is used to hash the password before storage.
 - The encrypted password is stored in the database, ensuring that it cannot be easily read even if accessed.

3.1.1.6 Admin Workflow Implementation

Use Case Diagram



Description & Priority

Incorporate administrative functionality for admin level users to allow them to manage users, courses etc.

Priority: Medium

Use Case

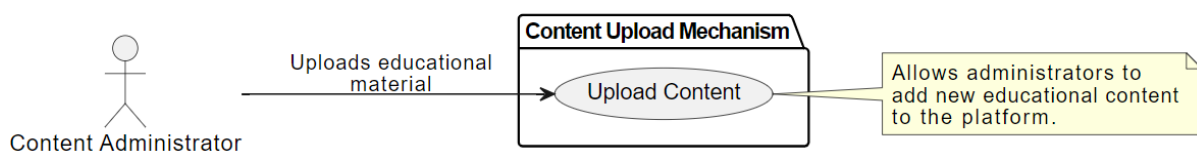
Scope: Administration of platform.

Flow Description:

- **Precondition:** Admin needs to manage users or content.
- **Activation:** Admin logs into the admin panel.
- **Main Flow:**
 - Admin uses the panel to remove or modify user profiles and/or add, remove or modify courses and content.
 - Content can be uploaded, removed, or updated through the panel.
 - Changes are logged.

3.1.1.7 Content Upload Mechanism

Use Case Diagram



Description & Priority

Implement a system for uploading educational content in PDF or XLSX formats.

Priority: Medium

Use Case

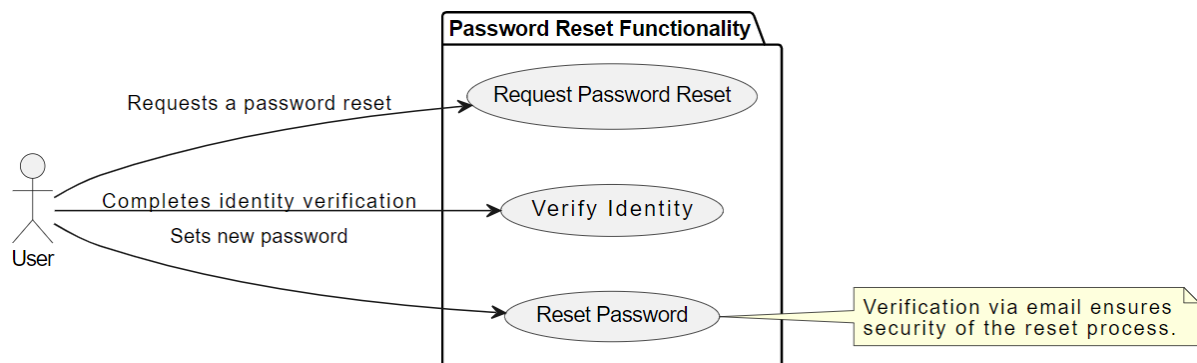
Scope: Content uploading.

Flow Description

- **Precondition:** New content needs to be added or existing content updated.
- **Activation:** An admin uploads new materials.
- **Main Flow:**
 - Files are uploaded through a secure interface to the server. XLSX file is parsed and sent to the Mongo DB with associated values to link to lesson.
 - Content is made available to users based on their enrolment status with the course in question.
 - Relevant database fields are updated.

3.1.1.8 Password Reset Functionality

Use Case Diagram



Description & Priority

Incorporate password reset functionality via a dedicated reset page. Temporary password is emailed to the email address associated with the user.

Priority: Medium

Use Case

Scope: Password reset process.

Flow Description:

- **Precondition:** A standard or admin level admin requests a password reset.
- **Activation:** The user selects the 'Forgot Password' option and enters their email address.
- **Main Flow:**
 - A password reset email is sent to the user's email address.
 - The user uses the new password emailed to them.
 - The new password is encrypted using bcrypt and updated in the database.

3.1.2 Data Requirements

The data requirements are as follows:

User profile data

- Name, email, password

Course data

- Name and description of the course

Content data

- Content name and description
- Linked course ID
- Content materials (in two forms: PDF and XLSX)

Mock email data

- Data used to populate mock emails to users in the email portal
- Held in MongoDB and parsed from XLSX to JSON

Content for five modules has been uploaded – this is in the form of PDF files and email collections in the MongoDB database. An XLSX template is supplied for email uploads (via a download button).

3.1.3 User Requirements

Users should be able to:

- Register and create profiles with secure authentication.
- Access a variety of interactive cybersecurity lessons.
- Participate in gamified modules with progressive difficulty levels (through content complexity).
- Track their progress and revisit completed lessons.

3.1.4 Environmental Requirements

The platform is designed as a web application so users will need web access and a browser to access the functionality. As it is web based, it is operating system agnostic so will run on any web enabled device, however, it is not designed to be accessed from mobile (although it is accessible, the styling will not appear as clean/optimised as the desktop browser version).

From a hosting perspective (as well as security) https is required so the app has been updated to use https and a self-signed certificate – in a real-life scenario a cname record would be added to the servers DNS records to accompany a third-party certificate. Both MSSQL and mongo have been installed locally on the server so there are less security considerations around the transport of data across distributed servers.

3.1.5 Usability Requirements

- Smooth user interface.
- Engaging content.
- Clear feedback with each user interaction.

3.2 Design & Architecture

The backend consists of node.js (express.js). The frontend consists of HTML, CSS, JavaScript and Bootstrap.

Database:

- Microsoft SQL server
- MongoDB

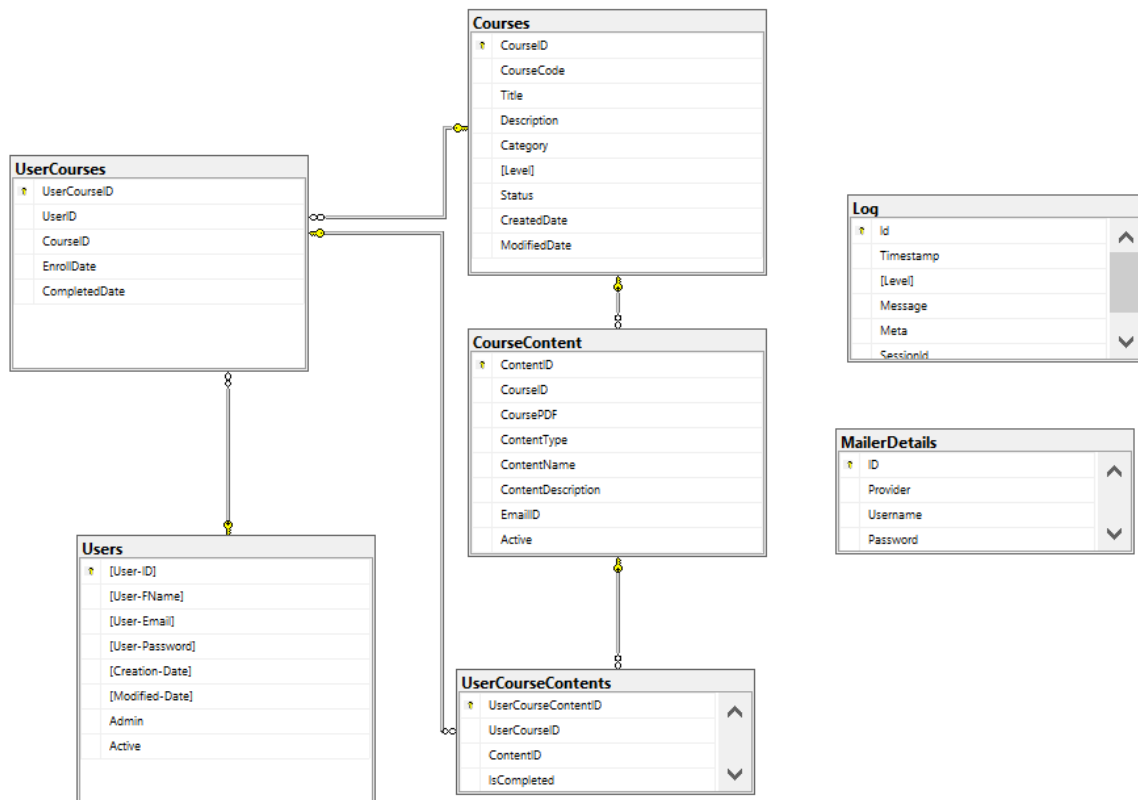
Middleware:

- Winston (logging)
- Helmet (security)
- csurf (security)
- bcrypt (security)




































Below are screenshots and images of the relevant designs and architecture of the system:

Database structure:
MSSQL

The database has the following schema:



All calls (with input values) are made to stored procedures from the backend to the database to increase security and have a central place to update queries/returned values etc.

- [-]  Stored Procedures
 - [-]  System Stored Procedures
 - [-]  **dbo.EnrollStudent**
 - [-]  dbo.PopulateUserCourseContents
 - [-]  dbo.sp_AdminAllCourses
 - [-]  dbo.sp_AllCourses
 - [-]  dbo.sp_DeleteCourseContent
 - [-]  dbo.sp_DeleteUserById
 - [-]  dbo.sp_FetchCourseContent
 - [-]  dbo.sp_FetchCourseContentById
 - [-]  dbo.sp_FetchCourseInfo
 - [-]  dbo.sp_FetchCoursePDF
 - [-]  dbo.sp_FetchEmails
 - [-]  dbo.sp_FetchUserById
 - [-]  dbo.sp_FetchUserCourseContent
 - [-]  dbo.sp_FetchUserData
 - [-]  dbo.sp_FetchUserGrades
 - [-]  dbo.sp_GetUserCourseID
 - [-]  dbo.sp_InsertCourse
 - [-]  dbo.sp_InsertCourseContent
 - [-]  dbo.sp_InsertUser
 - [-]  dbo.sp_IsEnrolled
 - [-]  dbo.sp_MarkContentComplete
 - [-]  dbo.sp_MyEnrolledCourses
 - [-]  dbo.sp_SoftDeleteCourse
 - [-]  dbo.sp_UpdateCourse
 - [-]  dbo.sp_UpdateCourseContent
 - [-]  dbo.sp_UpdateUserById
 - [-]  dbo.sp_UpdateUserEmail
 - [-]  dbo.sp_UpdateUserName
 - [-]  dbo.sp_UpdateUserPassword
 - [-]  dbo.sp_UserSearch
 - [-]  dbo.sp_ValidateUser
 - [-]  dbo.UpdateContentCompletion
 - [-]  dbo.UpdateCourseCompletion

Example of a stored procedure that updates several tables based on certain criteria (if a user registers for a course and new content is added – they should then be auto

enrolled):

```
ALTER PROCEDURE [dbo].[sp_InsertCourseContent]
    @CourseID INT,
    @CoursePDF NVARCHAR(MAX) = NULL,
    @ContentName NVARCHAR(255) = NULL,
    @ContentDescription NVARCHAR(MAX) = NULL,
    @ContentType NVARCHAR(50)
AS
BEGIN
    SET NOCOUNT ON;

    -- Insert the new course content
    INSERT INTO CourseContent (CourseID, CoursePDF, ContentName, ContentDescription, ContentType)
    VALUES (@CourseID, @CoursePDF, @ContentName, @ContentDescription, @ContentType);

    -- Get the newly inserted course content ID
    DECLARE @NewContentID INT = SCOPE_IDENTITY();

    -- Generate the new EmailID in the format CourseID_ContentID
    DECLARE @NewEmailID NVARCHAR(255);
    SET @NewEmailID = CAST(@CourseID AS NVARCHAR(50)) + '_' + CAST(@NewContentID AS NVARCHAR(50));

    -- Update the course content with the new EmailID
    UPDATE CourseContent
    SET EmailID = @NewEmailID
    WHERE ContentID = @NewContentID;

    -- Populate UserCourseContents table for all users enrolled in the course
    INSERT INTO UserCourseContents (UserCourseID, ContentID)
    SELECT UserCourseID, @NewContentID
    FROM UserCourses
    WHERE CourseID = @CourseID;

    -- Return the CourseID and the new ContentID
    SELECT @CourseID AS CourseID, @NewContentID AS ContentID;
END;
```

An example of the delete stored procedure – if there are tables with records linked to the content, a soft delete is performed (set to inactive instead):

```

ALTER PROCEDURE [dbo].[sp_DeleteCourseContent]
    @ContentID INT
AS
BEGIN
    BEGIN TRY
        -- Try to delete the record
        DELETE FROM CourseContent WHERE ContentID = @ContentID;
    END TRY
    BEGIN CATCH
        -- If delete fails, update the Active column to 0 (inactive)
        IF ERROR_NUMBER() = 547 -- Foreign key violation error number
        BEGIN
            UPDATE CourseContent
            SET Active = 0
            WHERE ContentID = @ContentID;
        END
        ELSE
        BEGIN
            -- Rethrow the error if it's not a foreign key violation
            THROW;
        END
    END CATCH
END

```

MongoDB

The mongo is titled emailDB and inside this database there are collections linked to the contentID and course ID from the MSSQL database. Each collection entry has the same structure:

```

_id: ObjectId('669bb98d894b61dcdcf1e1dc')
sender: "me@me.me"
snippet: "hello"
subject: "Important"
fake: false

```

Users upload collection content via an xlsx file with a fixed structure so that the correct field is mapped to the corresponding key for the collection i.e. column 1 links to the sender key, column 2 to the snippet key etc.

Node server endpoints

The frontend of the application calls the backend through its endpoints (depending on the request type – GET, POST, PUT or DELETE – a different endpoint is called with the required request content). If the request is successful a success request is sent to the

frontend to pass this to the user, and likewise if unsuccessful a negative response is sent to inform the user. Each endpoint has middleware protecting access and redirecting based on the results of the middleware checks (as already detailed in the Implementation section).

File structure

The root directory holds the express.js/node server file and the certificate information. There are also two log files generated to this area – error.log and combined.log. Both hold information generated by the logger object and this same information is also logged into the log table of the MSSQL database, so these are treated as backup logs. An env file holds the credentials for each interface/module that requires authentication (such as the DBs).

Inside the cricleCi folder my circle.yml file holds the instructions for deploying to the cloud server after each push to the main branch on GitHub.

Inside the public folder the HTML, CSS and JS files for the frontend are held. Inside the public folder there are two folders – assets and coursecontent. Assets holds logos and coursecontent holds uploaded PDF files that are stored there after upload for retrieval.

Frontend

Each page of the application has a corresponding JavaScript and HTML file. The css file is shared amongst all pages as well as a common.js file which handles logout requests. Many of the pages have html elements that are served via the js file as there is dynamic content – tables in particular:

```
public > JS grades.js > document.addEventListener("DOMContentLoaded") callback > courses.forEach() ca
1  document.addEventListener('DOMContentLoaded', async () => {
2      const gradesTableBody = document.querySelector('#grades-table tbody');
3
4      try {
5          const response = await fetch('/api/user-grades');
6          const courses = await response.json();
7
8          courses.forEach(course => {
9              const row = document.createElement('tr');
10             row.innerHTML = `
11                 <td>${course.Title}</td>
12                 <td>${course.Description}</td>
13                 <td>${course.Progress}%</td>
14             `;
15             gradesTableBody.appendChild(row);
16         });
17     } catch (error) {
18         console.error('Error fetching course grades:', error);
19     }
20 });
21
```

The html:

```
<div class="col-md-9">
  <div class="my-courses-section pt-5">
    <h2>My Course Progress</h2>
    <table class="table table-striped" id="grades-table">
      <thead class="thead-dark">
        <tr>
          <th scope="col">Course Title</th>
          <th scope="col">Description</th>
          <th scope="col">Progress</th>
        </tr>
      </thead>
      <tbody>
        <!-- Dynamic content will be injected here -->
      </tbody>
    </table>
  </div>
</div>
```

3.3 Implementation

Core functionalities are implemented using various algorithms and classes, such as:

Authentication and authorisation mechanisms to secure user data.

A middleware suite has been added to the node server to check for certain criteria to protect unauthorised access:

isAdmin checks the user record in the MSSQL database for the admin flag. If the user is an admin the access is allowed. This middleware is applied to admin-level pages/functionality. This functionality is also used to help the system server the correct user-level pages as the system differs depending on the user's privilege level.

```

function isAdmin(req, res, next) {
  if (req.isAdminChecked) {
    return next();
  }
  req.isAdminChecked = true;

  logger.info('Checking if request is from an admin');
  logger.debug('Session details:', { session: req.session });

  if (req.session.user && req.session.user.isAdmin) {
    logger.info('Admin verified, proceeding to next middleware');
    next();
  } else {
    logger.warn('Admin not verified, redirecting to login page', {
      sessionId: req.session.id,
      path: req.path,
      isAdmin: req.session.user ? req.session.user.isAdmin : 'undefined',
      user: req.session.user || 'undefined'
    });
    res.redirect('/login');
  }
}

```

isAdmin checks that the user has logged into the system before allowing them access to any page outside of the login and register page. This is applied to most endpoints.

```
// Authentication middlewares
function isAuthenticated(req, res, next) {
  if (req.isAuthenticatedChecked) {
    return next();
  }
  req.isAuthenticatedChecked = true;

  logger.info('Checking if request is authenticated');
  logger.debug('Session details:', { session: req.session });

  if (req.session.isAuthenticated) {
    logger.info('Request is authenticated, proceeding to next middleware');
    return next();
  } else {
    logger.warn('Request is not authenticated, redirecting to root', {
      sessionId: req.session.id,
      path: req.path
    });
    res.redirect('/');
  }
}
}
```

isEnrolled:

This blocks a user from accessing any course material they have not yet signed up for (the imagined scenario being someone manually changing the url of the site to redirect

to a different course):

```
//Enrolled check
async function isEnrolled(req, res, next) {
  const userId = req.session.user ? req.session.user.id : null;
  const courseId = req.params.courseId;
  logger.info(JSON.stringify(req.session));
  logger.info(`userId: ${userId}, courseId: ${courseId}`);

  if (!userId || !courseId) {
    logger.warn('User ID or Course ID not provided');
    return res.status(400).json({ message: 'User ID or Course ID not provided' });
  }

  try {
    let pool = await sql.connect(mssqlConfig);
    const request = pool.request()
      .input('UserID', sql.Int, userId)
      .input('CourseID', sql.Int, courseId);

    const result = await request.execute(`sp_IsEnrolled`);

    if (result.recordset.length > 0) {
      logger.info('User is enrolled in the course', { userId, courseId });
      next();
    } else {
      logger.info('User is not enrolled in the course', { userId, courseId });
      res.redirect('/courses');
    }
  } catch (err) {
    logger.error('Error checking enrollment:', err);
    res.status(500).json({ message: 'Server error' });
  }
}
```

csrf (an express middleware) has been used across the system to protect against unauthorised access or session spoofing by enforcing tokenised endpoint calls. Every frontend call to an endpoint (outside of a GET request) must carry a token.:

```
// API Routes - login user(API)
app.post('/login-user', csrfProtection, async (req, res) => {
  let email;
  logger.info('CSRF Token in POST:', req.body._csrf);
  try {
    const { email: reqEmail, password } = req.body;
    email = reqEmail;
```

```

fetch('/login-user', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'CSRF-Token': csrfToken // Include CSRF token in the headers
  },
  body: JSON.stringify(formData),
})
.then(response => {
  if (!response.ok) {
    throw new Error('Network response was not ok');
  }
  return response.json();
})
.then(data => {

```

Algorithms to simulate phishing attacks and gamified learning modules.

The js file get the emails from the email endpoint (that gets it from the mongo db):

```

async function fetchEmails(id) {
  try {
    const response = await fetch(`/emails/${id}`);
    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }
    let emailData = await response.json();
    // Sort and count fake and real emails from the fetched data
    emailData.forEach(email => {
      if (email.fake === true && fakeEmailCount < maxFakeEmails) {
        deliveredEmails.push(createEmail(email));
        fakeEmailCount++;
      } else if (email.fake === false && realEmailCount < maxRealEmails) {
        deliveredEmails.push(createEmail(email));
        realEmailCount++;
      }
    });

    //console.log(`Fetched emails - Fake: ${fakeEmailCount}, Real: ${realEmailCount}`);
    // Shuffle the emails to randomize the order of delivery
    shuffleArray(deliveredEmails);

    // Start the email delivery process
    startEmailDelivery();
  } catch (error) {
    console.error('Error fetching emails:', error);
  }
}

```

The emails are then delivered in set intervals to give the user time to process each:


```
// Function to deliver emails at intervals
function startEmailDelivery() {
  let emailDeliveryInterval = setInterval(function () {
    if (deliveredEmails.length > 0 && deliveryCount < maxEmailsDelivered) {
      let emailToDeliver = deliveredEmails.shift(); // Get the next email to deliver
      document.querySelector('.email-list').prepend(emailToDeliver);
      deliveryCount++;
      //console.log(`Delivered email count: ${deliveryCount}`);
    } else {
      clearInterval(emailDeliveryInterval); // Stop the interval after maxEmailsDelivered emails
    }
  }, 1500); // time interval setting
}
```

Based on the userLevel value (a randmoise number) the delivery ratio changes

```
document.addEventListener('DOMContentLoaded', function () {
  // Levels (the index) of real to fake emails
  const levelRatios = [
    {
      real: 9,
      fake: 1
    },
    {
      real: 8,
      fake: 2
    },
    {
      real: 7,
      fake: 3
    },
    {
      real: 6,
      fake: 4
    },
    {
      real: 5,
      fake: 5
    },
    {
      real: 5,
      fake: 5
    },
    {
      real: 4,
      fake: 6
    },
    {
      real: 7,
      fake: 3
    },
    {
      real: 1,
      fake: 9
    },
    {
      real: 5,
      fake: 5
    }
  ];

  function calculateEmailLimits(userLevel) {
    if (userLevel >= 1 && userLevel <= 10) {
      maxRealEmails = levelRatios[userLevel - 1].real;
      maxFakeEmails = levelRatios[userLevel - 1].fake;
    } else {
      // Default values
      maxRealEmails = 2;
      maxFakeEmails = 3;
    }
  }

  //log("Max real emails: ${maxRealEmails}, Max fake emails: ${maxFakeEmails}");
}
```

Data management classes to interface with SQL and NoSQL databases efficiently.

Each database has its own individual endpoints for CRUD operations but the link between them (in particular content creation) has the following flow and depends on

each step to be a successful creation of emails:

Emails are uploaded in xlsx format to the frontend which pares them into JSON format

and posts this to the content creation endpoint:

```
function submitCourseContent() {
  const courseId = window.location.pathname.split('/').pop();
  const contentName = document.getElementById('content-name').value;
  const contentDescription = document.getElementById('content-description').value;
  const pdfFile = document.getElementById('pdf-file').files[0];
  const xlsxFFile = document.getElementById('xlsx-file').files[0];

  if (!contentName || !contentDescription) {
    alert("All fields are required.");
    return;
  }

  const contentData = {
    contentName: contentName,
    contentDescription: contentDescription,
    contentType: ''
  };

  if (xlsxFFile) {
    const reader = new FileReader();
    reader.onload = (event) => {
      const data = new Uint8Array(event.target.result);
      const workbook = XLSX.read(data, { type: 'array' });
      const sheetName = workbook.SheetNames[0];
      const worksheet = workbook.Sheets[sheetName];
      const json = XLSX.utils.sheet_to_json(worksheet);

      contentData.xlsxData = JSON.stringify(json);
      contentData.contentType = 'Email';

      if (pdfFile) {
        uploadPDF(pdfFile, (filePath) => {
          contentData.pdfFile = filePath;
          sendCourseContent(courseId, contentData);
        });
      } else {
        sendCourseContent(courseId, contentData);
      }
    };
    reader.readAsArrayBuffer(xlsxFFile);
  } else if (pdfFile) {
    uploadPDF(pdfFile, (filePath) => {
      contentData.pdfFile = filePath;
      contentData.contentType = 'PDF';
      sendCourseContent(courseId, contentData);
    });
  } else {
    sendCourseContent(courseId, contentData);
  }
}
```

```

function sendCourseContent(courseId, contentData) {
  fetch('/csrf-token')
    .then(response => response.json())
    .then(data => {
      const csrfToken = data.csrfToken;

      fetch(`/api/course/${courseId}/content`, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          'CSRF-Token': csrfToken
        },
        body: JSON.stringify(contentData)
      })
        .then(response => {
          if (!response.ok) {
            throw new Error(`HTTP error! status: ${response.status}`);
          }
          return response.json();
        })
        .then(data => {
          if (data.success) {
            alert('Course content uploaded successfully');
            location.reload(); // Refresh the page
          } else {
            alert('Failed to upload course content: ' + data.message);
          }
        })
        .catch(error => {
          console.error('Error uploading course content:', error);
        });
    })
    .catch(error => {
      console.error('Error fetching CSRF token:', error);
    });
}

```

The endpoint creates the entry in the MSSQL database which generates a contentID which is then used as the key for creating the MongoDB entry:

```
//API - Create Course Content
app.post('/api/course/:courseId/content', isAuthenticated, isAdmin, async (req, res) => {
  const courseId = req.params.courseId;
  const { contentType } = req.body;

  const pdfFile = req.files ? req.files.pdfFile : null;
  const xlsxDat = req.body.xlsxDat ? JSON.parse(req.body.xlsxDat) : null;

  try {
    let pool = await sql.connect(mssqlConfig);

    const contentData = {
      coursePDF: pdfFile ? `${pdfFile.name}` : null,
      courseVideo: null,
      emailLevel: null,
      contentType: contentType
    };

    const result = await createCourseContent(courseId, contentData);

    if (xlsxDat) {
      await uploadEmails(result.collectionName, xlsxDat);
    }

    res.json({ success: true, contentId: result.contentId });
  } catch (err) {
    logger.error('Error occurred during course content upload:', err);
    res.status(500).json({ message: 'Server error' });
  }
});
```

```
// Helper function to create course content
async function createCourseContent(courseId, content) {
  try {
    let pool = await sql.connect(mssqlConfig);
    const request = pool.request()
      .input('CourseID', sql.Int, courseId)
      .input('CoursePDF', sql.NVarChar(sql.MAX), content.coursePDF || null)
      .input('ContentName', sql.NVarChar(255), content.contentName || null)
      .input('ContentDescription', sql.NVarChar(sql.MAX), content.contentDescription || null)
      .input('ContentType', sql.NVarChar(50), content.contentType || null);

    const result = await request.execute('sp_InsertCourseContent');

    if (result.recordset.length > 0 && result.recordset[0].ContentID) {
      const contentId = result.recordset[0].ContentID;
      const collectionName = `${courseId}_${contentId}`;

      logger.info('Course content created successfully: CourseID: ${courseId}, ContentID: ${contentId}');

      return { courseId, contentId, collectionName };
    } else {
      throw new Error('Failed to create course content.');
```

Note the returned values which are then sent to the uploadEmails function

```
// Helper function to upload emails
async function uploadEmails(collectionName, emailData) {
  try {
    const collection = client.db("emailDB").collection(collectionName);
    const result = await collection.insertMany(emailData);

    logger.info(`Emails uploaded successfully: CollectionName: ${collectionName}, InsertedCount: ${result.insertedCount}`);

    return {
      success: true,
      insertedCount: result.insertedCount,
      insertedIds: result.insertedIds
    };
  } catch (err) {
    logger.error(`Failed to insert emails: ${err.message}, CollectionName: ${collectionName}`);
    throw err;
  }
}
```

When retrieving the emails the Email field of the Coursecontent table is referenced to get the correct collection name for the MongoDB database:

	ContentID	CourseID	CoursePDF	ContentType	ContentName	ContentDescription	EmailID	Active
1	47	1	/coursecontent/diagram.pdf	Email	c	c	1_47	1

The screenshot shows the MongoDB Compass interface. On the left, the 'Databases' sidebar is expanded to show 'emailDB', which contains a collection named '1_47'. The '1_47' collection is highlighted with a red box. The main panel displays the 'Documents' tab for 'emailDB.1_47'. It shows a list of documents with fields: _id, sender, snippet, subject, and fake. The first document is highlighted, showing its details.

3.4 Graphical User Interface (GUI)

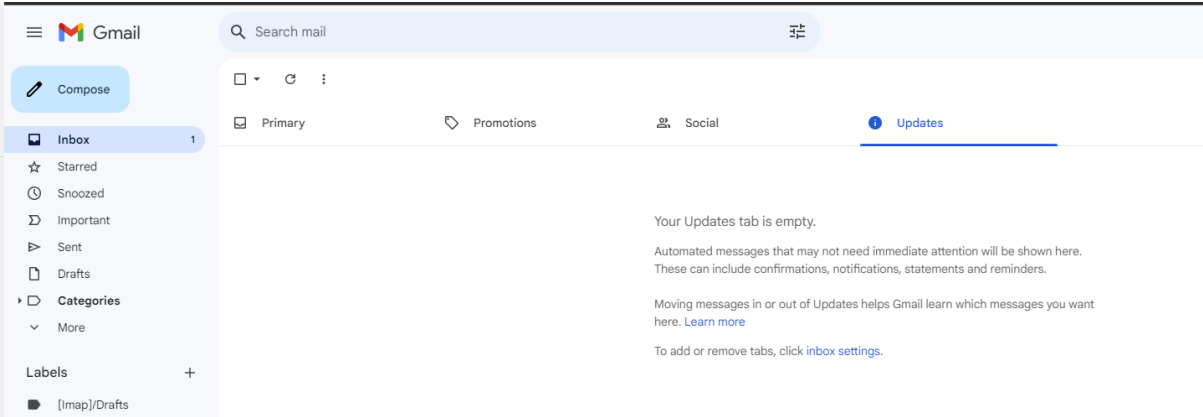
The standard platform (non-admin users) features a welcome dashboard, a menu bar with links for easy access to different modules, a progress tracking section, and courses designed with a lifelike email portal, a user settings page and (if an admin) there is both a user and course management section. The email portal was modelled on Gmail.

Below are the relevant screenshots and images from each section:

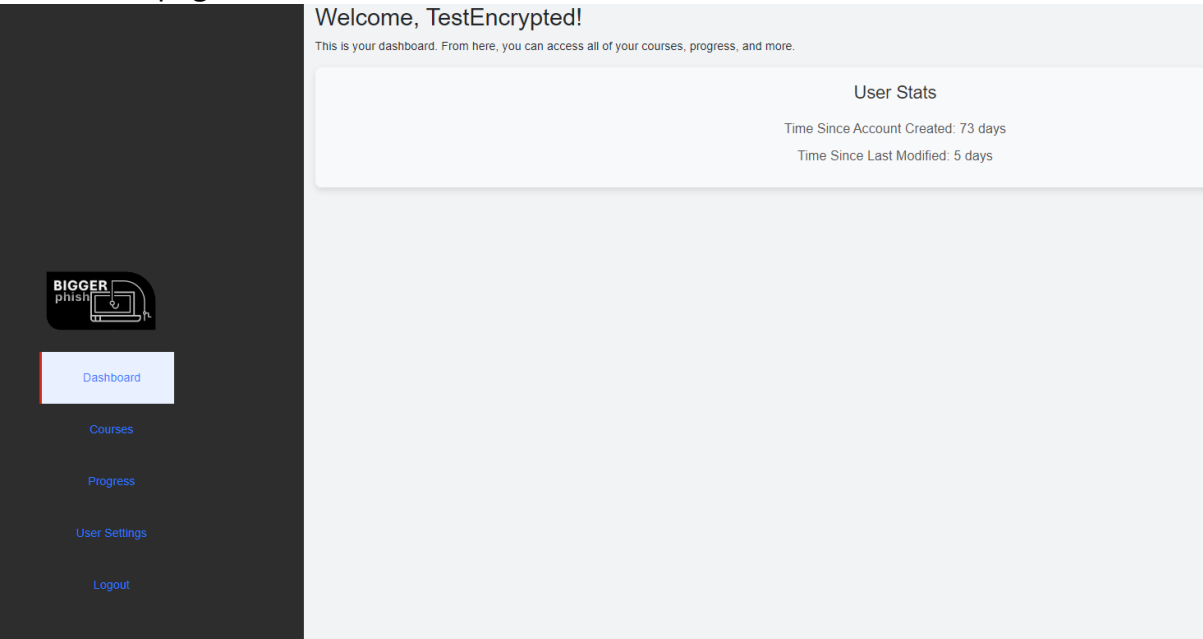
Email portal:




Gmail:



User home page



User courses page:



Dashboard

Courses

Progress

User Settings

Logout

My Courses

Digital Evidence
Test Course
Go to Course

TestNew
TEstNew
Go to Course

FirstUpload
testingxlsxParse
Go to Course

secondUpload
testingupload
Go to Course

All Courses

Digital Evidence
Test Course
Go to Course

TestNew
TEstNew
Go to Course


FirstUpload
testingxlsxParse
Go to Course

5th
5
Enroll

secondUpload
testingupload
Go to Course

hello
hello
Enroll

User progress page:



Dashboard

Courses

Progress


User Settings

Logout

My Course Progress

Course Title	Description	Progress
Digital Evidence	Test Course	100%
FirstUpload	testingxlsxParse	100%
secondUpload	testingupload	0%
TEstNew	TEstNew	0%

Admin or User Settings page:



Dashboard

Courses

Progress

User Settings

Logout

User Settings

Update Details

Email Address

testEncrypted@email.com

Update Email

Name


TestEncrypted

Update Name

Update Password

Request Account Deletion

Admin course management page:



Dashboard

Course Management

User Management

User Settings

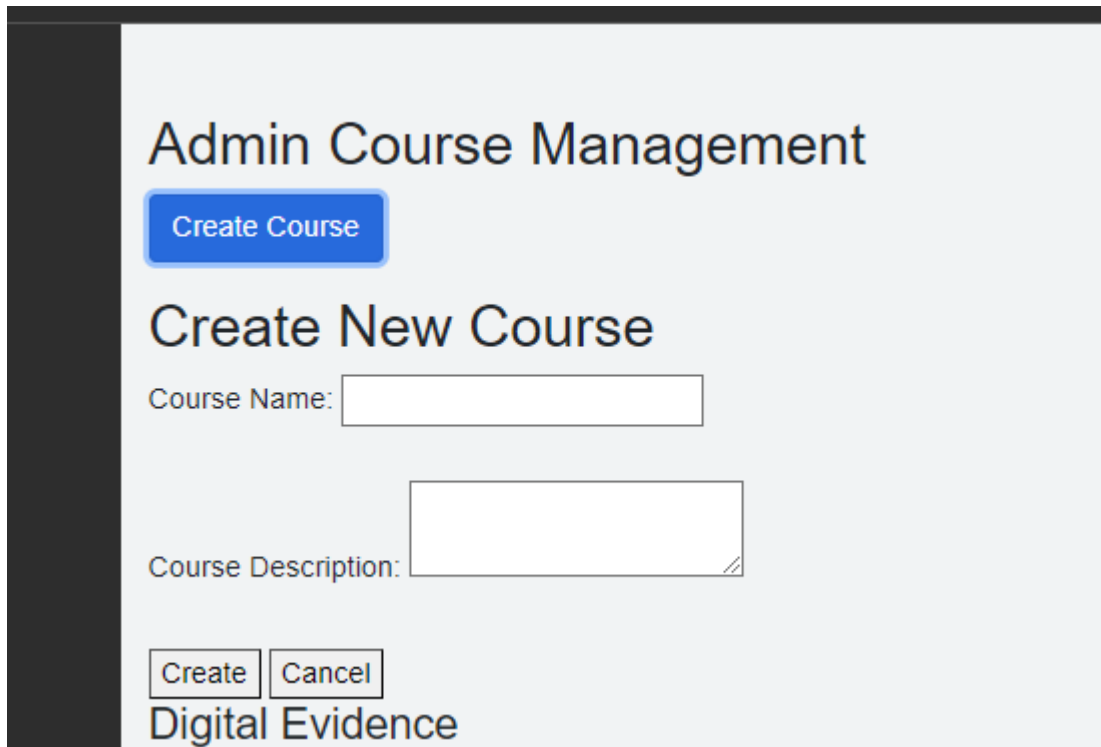
Logout

Admin Course Management

Create Course

Digital Evidence	FirstUpload	secondUpload
Test Course	testingxlsxParse	testingupload
<div>EditDelete</div>	<div>EditDelete</div>	<div>EditDelete</div>
TEstNew	5th	hello
TEstNew	5	hello
<div>EditDelete</div>	<div>EditDelete</div>	<div>EditDelete</div>

Clicking the buttons will either trigger functionality for deletion or open a new page or modal:



Admin Course Management

[Create Course](#)

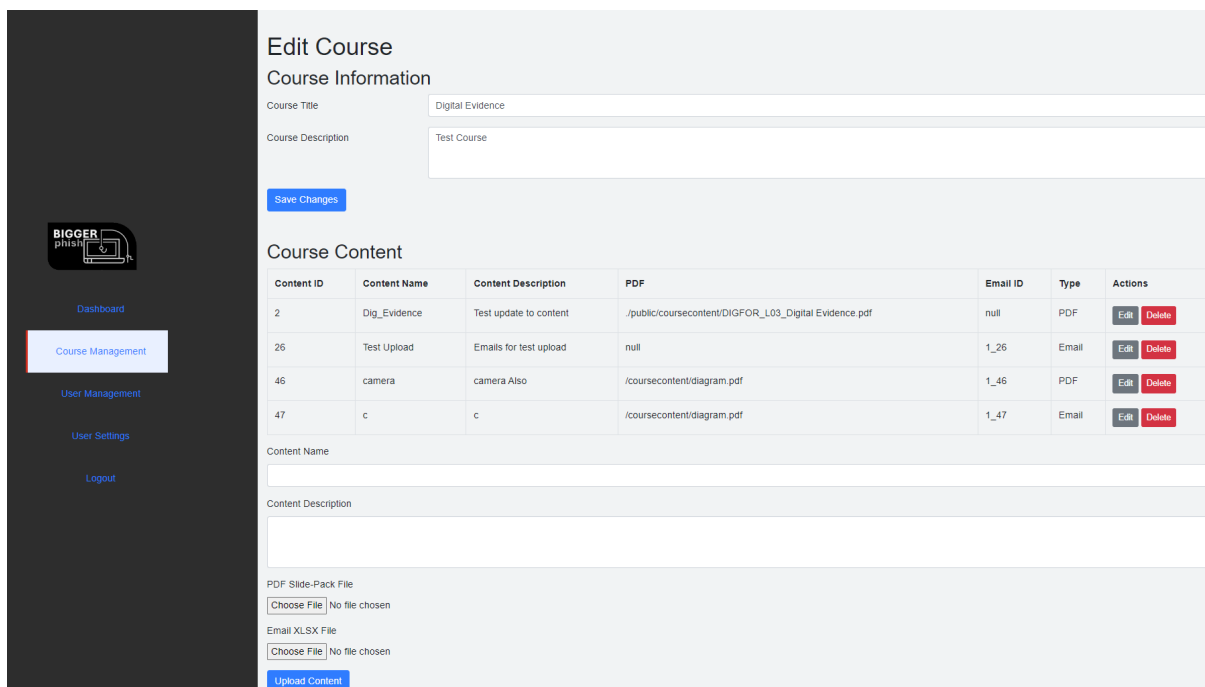
Create New Course

Course Name:

Course Description:

[Create](#) [Cancel](#)

Digital Evidence



Edit Course

Course Information

Course Title:

Course Description:

[Save Changes](#)

Course Content

Content ID	Content Name	Content Description	PDF	Email ID	Type	Actions
2	Dig_Evidence	Test update to content	/public/coursecontent/DIGFOR_L03_Digital Evidence.pdf	null	PDF	Edit Delete
26	Test Upload	Emails for test upload	null	1_26	Email	Edit Delete
46	camera	camera Also	/coursecontent/diagram.pdf	1_46	PDF	Edit Delete
47	c	c	/coursecontent/diagram.pdf	1_47	Email	Edit Delete


Content Name:

Content Description:

PDF Slide-Pack File: [Choose File](#) No file chosen


Email XLSX File: [Choose File](#) No file chosen

[Upload Content](#)



- Dashboard
- Course Management**
- User Management
- User Settings
- Logout

Admin user management page:



Dashboard

Course Management

User Management

User Settings

Logout

Admin User Management

PadraigEncrypted

Email: pEncrypted@email.com

Edit

Delete

John

Email: john@example.com

Edit

Delete

TestEncrypted

Email: testEncrypted@email.com

Edit

Delete


Padraig

Email: pmickeyc@gmail.com

Edit

Delete

Home page:



Home

Welcome to BiggerPhish

Your central hub for cyber security education.

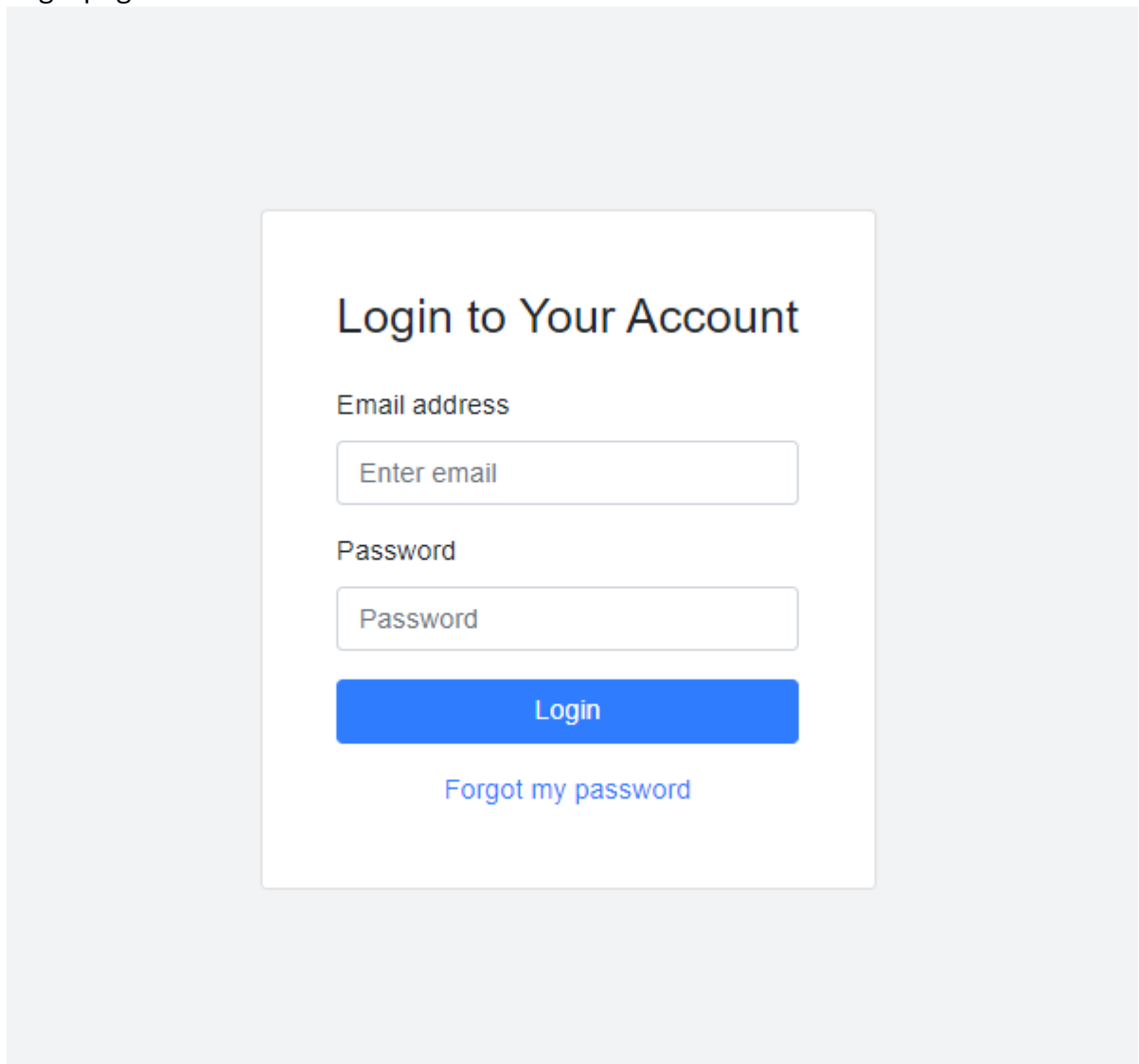
Login

Register

Learn anytime, anywhere. Enhance your cyber security skills with our self-directed learning modules.

34

Login page:

A login page with a light gray background. In the center is a white rectangular box with rounded corners. Inside this box, the title "Login to Your Account" is centered at the top in a large, bold, black font. Below the title, the text "Email address" is left-aligned in a smaller black font. Underneath is a white input field with rounded corners and a thin gray border, containing the placeholder text "Enter email". Below this, the text "Password" is left-aligned in a smaller black font. Underneath is another white input field with rounded corners and a thin gray border, containing the placeholder text "Password". Below the password field is a solid blue rectangular button with rounded corners, containing the text "Login" in white. At the bottom of the white box, the text "Forgot my password" is centered in a blue, underlined font.

Login to Your Account

Email address

Password

Login

[Forgot my password](#)

Register page:

Create an Account

Full Name

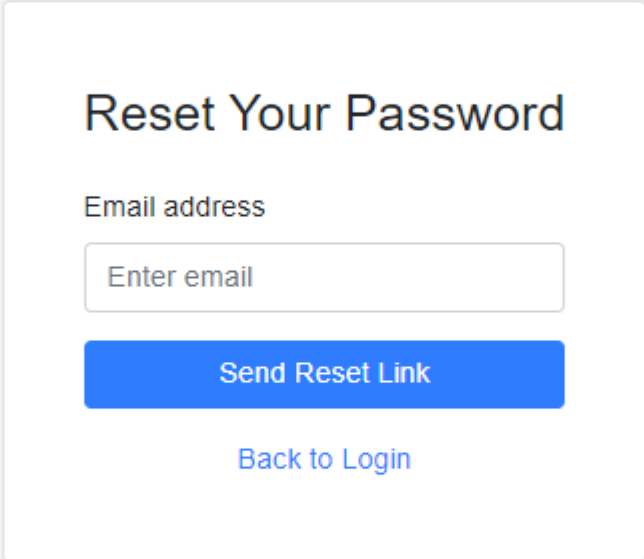
Email address

Password

Confirm Password

Register

Forgot password page:



The image shows a 'Reset Your Password' form centered on a light gray background. The form is a white rectangle with a thin gray border. At the top, the title 'Reset Your Password' is displayed in a large, bold, black font. Below the title, the label 'Email address' is shown in a smaller, regular black font. Underneath the label is a text input field with a light gray border and the placeholder text 'Enter email' in a light gray font. Below the input field is a solid blue button with the text 'Send Reset Link' in white. At the bottom of the form, the text 'Back to Login' is displayed in a blue, underlined font, serving as a link.

3.5 Testing

Unit Testing

Jest is used to perform unit testing on the server.js file endpoints in an automated fashion (server.test.js). An example of the output from the tests:

Educational Platform API Tests

```
✓ should register a new user (274 ms)
✗ should login an existing user (274 ms)
✓ should delete the user (386 ms)
✓ should get the root HTML file (511 ms)
✗ should fetch user data for logged-in user (242 ms)
✗ should serve the user HTML page (247 ms)
✓ should serve the admin HTML page (1183 ms)
✓ should serve the admin courses management page (902 ms)
✓ should serve the admin users management page (922 ms)
✓ should retrieve all courses for admin (1070 ms)
✓ should retrieve all users for admin (1441 ms)
✗ should serve the course-specific HTML page (357 ms)
✓ should retrieve and serve the PDF content for the given course (459 ms)
✓ should retrieve the courses that the logged-in user is enrolled in (382 ms)
✓ should serve the email HTML page (296 ms)
✓ should retrieve all emails from the specified collection (283 ms)
✓ should serve the courses HTML page (171 ms)
✓ should serve the login HTML page (85 ms)
✓ should serve the registration HTML page (115 ms)
✓ should serve the forgotten password HTML page (117 ms)
✓ should serve the settings HTML page based on user role (230 ms)
✓ should provide a CSRF token (81 ms)
```

From these I can determine if individual endpoints are functioning as expected, in particular after any changes are made. An example of a more granular log message on a failed test:

```
• Educational Platform API Tests > should login an existing user

expect(received).toBe(expected) // Object.is equality

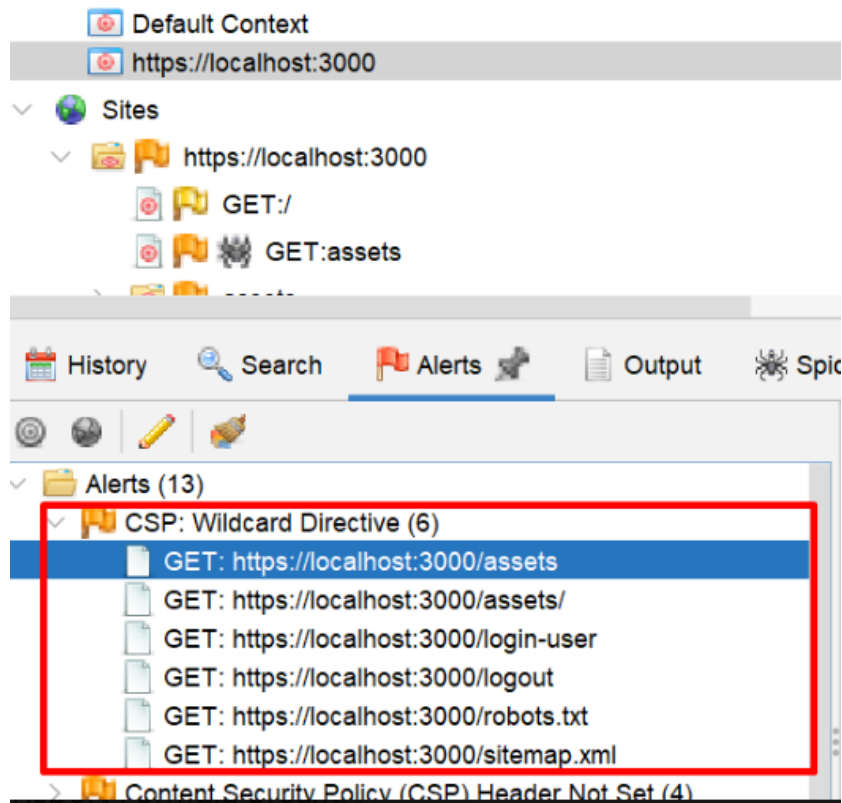
Expected: 200
Received: 401

  92 |         const response = await agent.post('/login-user').set('CSRF-Token', csrfToken).send(loginUser);
  93 |         console.log("Login response:", response.body);
> 94 |         expect(response.status).toBe(200);
     |                                   ^
  95 |         expect(response.body.success).toBe(true);
  96 |     });
  97 |

    at Object.toBe (server.test.js:94:33)
```

Security Testing

Spider attacks were used to test the security aspects of the functionality via OWASP ZAP. Credentials were added to the parameters of ZAP along with contexts to show a logged in or logged out state for the site. With these in place automated attacks could be run and amendments then made to the security aspects of the site based on the alerts – an example of the alerts flagged by the attacks:



As a result of the ZAP security tests, the following aspects were added to the platform:

- Content headers
- Returned values cleaned up
- Console logs cleaned up
- Tokenised calls to the frontend
- HTTPS enabled and configured
- Cookies enabled and configured

Functional Testing

Requirement	Action	Expected Result	Result
Participation in simulated Phishing attacks	Present user with five simulated phishing attack scenarios. User identifies each as attack or safe.	System displays the user's score and allows the user to proceed or repeat the scenario.	Pass
Time Bound Emails Validation	Present user with a mixture of emails to categorize within an allocated time. Level	User successfully categorizes emails within the time limit, and difficulty	Pass

	progresses from easy to hard.	increases with each level.	
User Profile Handling	Save user profile data from the database.	User profile data is accurately stored without errors.	Pass
User Profile Handling	Update user profile data from the database.	User profile data is accurately updated without errors.	Pass
User Profile Handling	Retrieve user profile data from the database.	User profile data is accurately retrieved without errors.	Pass
User Profile Handling	Delete user profile data from the database.	User profile data is accurately deleted without errors.	Pass
Admin Workflow	Provide admin with permissions for user management.	Admin can manage users without error.	Pass
Admin Workflow	Provide admin with permissions for content management.	Admin can manage content without error.	Pass
Efficient Content Management	Upload files securely to the server, parse files appropriately, and update database fields.	Files are uploaded securely, parsed correctly, and database is updated.	Pass
Credentials Management	Store credentials in environment variables. Securely access credentials while connecting to the database.	Credentials are securely stored and accessed.	Pass
Security of User Authentication Data	Encrypt passwords using Bcrypt and store them in the database.	Passwords are securely encrypted and stored in the database.	Pass
DB Content Management	Upload new content in xlsx or PDF formats.	New content is uploaded, databases are updated, and content is accessible to enrolled users.	Pass
Reset Password Functionality	Send reset password email to users.	Users receive reset password emails, and new passwords are successfully saved.	Pass

Usability Testing	Navigate the application as both a standard and admin user.	Users can navigate the application easily without confusion.	Pass
Content Testing	Create and upload content that is accurate and progressively difficult.	Content is relevant and has a progressive difficulty curve.	Pass

3.6 Evaluation

The evaluation assessed the performance, security and overall functional completeness of the platform. Google Lighthouse was used to evaluate the performance, ZAP was used to evaluate the security (along with the results of the testing suite performed as mentioned above), and the defined functional requirements as specified by the use cases dictated the functional completeness of the implementation (assessed through manual testing).

Performance

Google Lighthouse was used to assess the platform's performance. The key metrics and results are as follows:

- Overall Performance Score: 100
- First contentful paint: 0.6 seconds
- Largest contentful paint: 0.6 seconds
- Total blocking time: 0 ms
- Cumulative layout shift: 0
- Speed index: 0.6 seconds

The above results indicate that the platform performs at above average speed. Multimedia, expensive animation and styling etc. were kept to a minimum to ensure site speeds were as high as possible.

Security

Security was evaluated with ZAP to gather an overall sense of the secureness of the platform. Through spider attacks, the number of alerts returned (and their severity as defined by the vulnerabilities CVE record) was used as the guiding metric to evaluate the security. Alerts were dealt with based on their severity.

Currently the testing returns two 'level medium' alerts and seven 'low level' alerts, all of which are connected to components unrelated to sensitive data (and as such deemed acceptable).

System Functionality

The system functionality was tested using the methods outlined in the testing section above.

The results from the testing provided guidance on the completeness of the implementation. The results were gathered from the number of expected outcomes against the number of tests in comparison with the number of unexpected outcomes.

Adjustments were made based on the results to mitigate unexpected results.

4. Conclusions

In conclusion, the project's experience has been one of great personal growth. A lot has been learned about the languages used – node.js in particular. The project has given me confidence to approach a new project with a node.js application with an assurance that the results would be good, and the workload would be manageable.

In assessing the implementation of this project, the strengths are:

- The dual database implementation works effectively.
- The link to SMTP server for emailing is simple and works reliably.
- The parsing of the xlsx files to create the JSON format for creating email database collections is consistent.
- The algorithm for selecting and delivering the mock emails to users was a personal highlight that I believe is well implemented.

In contrast, I believe the limitations are:

- A plain/no frills front end that could have been given more vibrant and animated styling.
- A simplified content delivery system – more multimedia content would have improved the thoroughness of the platform as a learning tool.

The overall completeness of the implementation is in line with the functional requirements as set out in the above report section, however the limitation detailed above is in comparison to a real world application of a similar type. Overall, I think the platform is effective as a simple learning tool for phishing attack education – it is focused on a singular purpose. The next section details how further amendments could be made to improve the platform.

5. Further Development or Research

Given how much I learned over the course of the project, I envision a great deal of refactoring of code and reimplementing of functionality. I would separate my server.js into distinct files for better management of each component. I would like to explore a more refined method of templating html so that there is a single point for updating that would carry across all pages based on their class/category (user vs admin user being one example). I would explore more interactive media types such as video and video

game type lessons – the main reason I avoided using video for this project was for storage reasons, I did not want hosting costs to grow, else a video player would have been implemented also with videos stored on an S3 bucket and retrieved that way.

I believe I would enhance the user interface to have more smooth animations and reactive elements, as well as incorporating a sound library to the email portal for more instant feedback on interactions with the emails.

Finally, I would research more learning science elements to deliver more effective content and ensure it is being served the correct manner for maximum retention for the users.

6. References

- Mozilla (2023) 'PDF.js Examples', [online] Available at: <https://mozilla.github.io/pdf.js/examples/> (Accessed: 10 March 2024).
- Stack Overflow (2023) 'Bootstrap Modal Interaction Issue', [online] Available at: <https://stackoverflow.com/questions/41292673/bootstrap-modal-opens-but-stays-in-gray-background-and-cannot-close-or-interact> (Accessed: 25 April 2024).
- Syed, B. (2014) Beginning Node.js. Apress.
- Powers, S. (2016) Learning Node: Moving to the Server-Side. 2nd edn. O'Reilly Media, Inc.
- Hahn, E. (2016) Express in Action: Writing, building, and testing Node.js applications. Simon and Schuster.
- Oakley, B., Sejnowski, T., & McConville, A. (2024) Learning How to Learn. Bentang Pustaka.
- Oles, N. (2023) 'How to Catch a Phish', in Phishing Tactics and Techniques. Springer Nature.
- Hoffman, A. (2020) Web Application Security: Exploitation and Countermeasures for Modern Web Applications. O'Reilly Media, Inc.
- Zalewski, M. (2011) The Tangled Web: A Guide to Securing Modern Web Applications. No Starch Press.
- Crockford, D. (2008) JavaScript: The Good Parts. O'Reilly Media, Inc.
- MetaCompliance (2023) 'Ultimate Guide to Phishing', [online] Available at: <https://www.metacompliance.com/lp/ultimate-guide-phishing> (Accessed: 2 February 2024).
- DigitalOcean (2023) 'SSH Essentials: Working with SSH Servers, Clients, and Keys', [online] Available at: <https://www.digitalocean.com/community/tutorials/ssh-essentials-working-with-ssh-servers-clients-and-keys> (Accessed: 7 May 2024).
- CircleCI (2023) 'Deploy Over SSH', [online] Available at: <https://circleci.com/docs/deploy-over-ssh/> (Accessed: 30 June 2024).
- Baeldung (2023) 'OpenSSL Self-Signed Cert', [online] Available at: <https://www.baeldung.com/openssl-self-signed-cert> (Accessed: 15 January 2024).

- MongoDB (2023) 'Develop Applications with MongoDB', [online] Available at: <https://www.mongodb.com/docs/develop-applications/> (Accessed: 4 April 2024).
- Noah (2023) 'Bcrypt: A Beginner's Guide', Medium. [online] Available at: <https://medium.com/@CodeNameNoah/bcrypt-a-beginners-guide-e2293cc1eeb6> (Accessed: 1 July 2024).
- Express.js (2023) 'CSURF Middleware', [online] Available at: <https://www.expressjs.com.cn/resources/middleware/csrf.html> (Accessed: 15 May 2024).
- Codino (2023) 'Secure Your Express.js App with Helmet.js', Medium. [online] Available at: <https://codino.medium.com/secure-your-express-js-app-with-helmet-js-a-step-by-step-guide-632b7d94da78> (Accessed: 20 February 2024).
- LogRocket (2023) 'Using Helmet in Node.js to Secure Your Application', [online] Available at: <https://blog.logrocket.com/using-helmet-node-js-secure-application/> (Accessed: 28 March 2024).
- Jest (2023) 'Getting Started with Jest', [online] Available at: <https://jestjs.io/docs/getting-started> (Accessed: 12 April 2024).
- SheetJS (2023) 'SheetJS Utilities', [online] Available at: <https://docs.sheetjs.com/docs/api/utilities/> (Accessed: 5 June 2024).
- NPM (2023) 'Express File Upload', [online] Available at: <https://www.npmjs.com/package/express-fileupload> (Accessed: 18 January 2024).
- NPM (2023) 'UUID v4 Package', [online] Available at: <https://www.npmjs.com/package/uuidv4> (Accessed: 10 February 2024).

7. Appendices

Project Proposal



National College of Ireland

Project Proposal

Title: BiggerPhish: An Interactive Cybersecurity Training Platform

Date: [Insert Date]

Programme Name: BSc in Computing

Specialisation: Cybersecurity

Academic Year: 2023/2024

Student Name: Padraig McCauley

Student Number: 20123744

Student Email: x20123744@student.ncirl.ie

Objectives

Established resources, including WebGoat by OWASP and the game "Papers, Please," serve as the foundation for BiggerPhish—a captivating, informative cybersecurity training platform. The project aspires to use these resources to create the platform on which the real teaching can happen. Either through WebGoat or "Papers, Please," I can achieve a lesson on key cybersecurity issues (phishing, for instance) that the user can interact with and thereby learn more effectively than if they simply read about the issue.

Using "Papers, Please" as a foundation, I can apply the game mechanics that make the simulation more engaging to our email phishing training module. "Papers, Please" is a game about decision-making and time management—the two aspects of gameplay that I believe most directly translate to what I want our users to do within the phishing simulation and why I want them to do it. The authors of the game have done a great job of making the scenarios challenging and somewhat life-like. And they have endowed those scenarios with an increasing level of difficulty that, once grasped, equips the player to face a similar but potentially more severe situation in real life.

Demonstrating technical skills: Show individual technical skills through the exercise using both SQL and NoSQL databases in a dual-database environment. This guarantees the correct handling of user profiles, the accurate tracking of user progress, and dynamic content for the different modules. The Learning Experience: Ensure that this platform not only educates but also is accessible, in an environmentally friendly way, for users with differing sets of technical backgrounds. The aim is to create a learning platform that is a true challenge for advanced users yet a friendly space for beginners.

Background to BiggerPhish

BiggerPhish is, in part, a response to the demand for better cybersecurity education as cyber threats continue to rise. BiggerPhish is inspired by the likes of WebGoat by OWASP but ultimately intends to provide a significantly more advanced experiential learning platform over and above what current resources provide. WebGoat already creates a great model for building an interactive approach to security education, focused on web application vulnerabilities.

The second source of inspiration for the project is the deeply immersive and decision-intensive gameplay in "Papers, Please," a game that blends critical thinking and detail orientation under significant time pressure. It is this idiosyncratic engagement with the user in complex tasks within a pressured time frame that BiggerPhish emulates in its email simulation platform for various cybersecurity threats that users must recognize and respond to quickly.

With all the educational content from WebGoat combined with the efficient, time-pressured decision-making in "Papers, Please," BiggerPhish will deliver an interactive learning experience that's engaging. The addition of a dual database system, using SQL and NoSQL, is a personal challenge aimed at demonstrating advanced technical proficiency and innovative problem-solving in database management.

State of the Art

The field of cybersecurity training platforms has seen tremendous growth, with several solutions being developed to satiate a growing demand for hands-on cyber defense skills. A typical example in this area is WebGoat from OWASP (Open Web Application Security Project), used both as a benchmark and source of ideas in the development of BiggerPhish.

Some commonalities with WebGoat:

- Like WebGoat, BiggerPhish focuses on an interactive approach to learning. Both platforms engage the user through interactive activities where the user gets to apply what they have learned in a simulated environment.
- In the same way that WebGoat does, BiggerPhish provides a way to practice realistic cybersecurity problems. This basically makes sure that the skills acquired are provable towards real life.
- Both platforms seek to touch upon a broad spectrum of topics in cybersecurity, making sure their users are universally educated and prepared.

Differences from WebGoat:

- BiggerPhish introduces a new gamified approach to identifying phishing and malicious emails. This component, borrowing gameplay from "Papers, Please," differs from the lesson structure of WebGoat, therefore being more immersive and engaging.
- The time-based email simulation, slowly increasing in difficulty as the user goes on, creates an urgency and stress that can rarely, if ever, be felt in WebGoat. This was designed to ensure the user is best equipped to get used to the speed with which threats are typically orchestrated in reality.
- An additional piece of complexity added to BiggerPhish is the technical challenge in integrating SQL and NoSQL databases in handling different aspects of the

platform. This does not only demonstrate more technical skills and sophistication but complexity, both in data management and security.

- WebGoat functions very effectively as training for a person who is experienced in security. BiggerPhish will be designed in such a way that it can be used even by persons who are not very conversant with the art of cybersecurity. The objective here is to reduce the barrier of entry of cybersecurity education, reducing fear to a low level for novices.

The development of BiggerPhish will start with an exhaustive phase for identifying requirements. My plan is to work closely with stakeholders to conduct both interviews and surveys. These will engage stakeholders at all levels and in all locations—internationally and domestically—to capture an accurate and thorough set of requirements. The scoped set of functional and non-functional requirements will then be translated into the Agile methodology that I will use to develop the system. I will work at an iterative pace, breaking down the project into two-week sprints. The Agile methodology is employed mostly for the purpose of flexibility and adaptability. If I need to make mid-course corrections based on stakeholder feedback or new project insights, I will be able to do so using this approach.

In terms of the technology stack, I have selected Express.js for the backend, taking advantage of its efficient and scalable nature. The frontend is going to be artistry with HTML5, CSS3, and JavaScript, providing a vivid and engaging user experience. To Bootstrap, I will also apply my reasons for the mobile-first, responsive design capabilities of this framework. A major part of my reason for using Bootstrap is its ease of use, and its compatibility with the design aesthetic I am striving for. I might also tell you about the reasons for using a reactive library like Vue.js or React.js for the framework of this application, but that will be another story.

The development approach will be inclusive of unit and integration testing done to high standards to ensure code quality and basic functionality. I plan to use Git as a version control system to keep things tidy, and I will use a web-based hosting service such as GitHub or Bitbucket to further enhance collaboration and ensure that the code remains clean and maintainable. Code that is written for a cybersecurity application like BiggerPhish must comply with certain security-related coding standards; otherwise, the application will be more susceptible to penetration and attack. To help preempt such vulnerabilities, I plan to do code audits as well as penetration tests of the application's functionality.

To achieve consistent feature delivery alongside application stability and security, I will establish a CI/CD pipeline for deployment.

I will develop BiggerPhish using Express JS for the server-side framework and JavaScript, HTML, and CSS for the client-side interface. I will utilize a dual database system; I will use an SQL database for user data and a NoSQL database for content. I will focus on interaction security and efficient data handling and storage. I require access to a web server (and cloud database) for development and testing. Our project

plan outlines our steps from initial design to final testing and includes interface development, database integration, server-side scripting, and user testing.

The project will move forward in two-week increments, with clearly defined goals for each period. The first two "sprints" will focus on design and prototyping. User research will be done to inform the design decisions. Initial wireframes, mockups, and an interface prototype will be created. In the next two sprints, I will set up a backend server in an environment suitable for iterative development; a CI/CD (continuous integration/continuous deployment) pipeline will be established; and basic security features will be implemented.

The database schema will be constructed and integrated, the principal features will be realized in modules, and the authentication and authorization mechanisms will be implemented during the database integration and module development phase (Sprints 5-8). The user interface development phase (Sprints 9-12) will concentrate on fine-tuning the user interface in light of initial prototype feedback; implementing front-end functionality; and integrating the front-end with back-end services. The syllabus creation module will be designed and developed in Sprints 13-14, and then tested for usability and functionality.

The phase of testing and refining the system (Sprints 15-22) will gathering of requirement test results, using which to refine features and interfaces of the system, doing security testing and hardening of the system, and a few other things to ensure that the system is as effective and usable as it can be. The final phase (Sprints 23-24) will consist of end-to-end testing of the whole BiggerPhish system and a final review of documentation and support materials. The platform will be tested not just for functionality (and, in doing so, it will be checked against UI best practices), but also for security, with a combination of tests to ensure that the system is safe.

Reflective Journals

Supervision & Reflection Template

Student Name	Padraig McCauley
Student Number	20123744
Course	BSCCYBE4
Supervisor	

Month: November

What?

Reflect on what has happened in your project this month?

This month I linked in with my supervisor. I had originally been given someone different but this was changed after my initial contact. My new supervisor and I met twice to discuss my original idea and some suggestions were given to adjust the project to be more in line with the cybersecurity field. The points I had around web technology and using two types of databases were kept and moving the subject matter and focus towards cybersec training has been the focus instead. I have been researching educational sites for cybersecurity with an interest in anything interactive/gamified. WebGoat has been of particular interest.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

Moving to a more cybersecurity focus project means I can start to incorporate more directly the lessons I am learning in my modules this semester which I am happy about. I feel this will allow me to better internalise the material and applying this knowledge directly into a project I can later showcase to employers is a great way to kill two birds with one stone. The suggestion was a good pointer from my supervisor and I feel we have begun a productive relationship.

Now What?

What can you do to address outstanding challenges?

I will need to form a more considered idea for my project ahead of the proposal submission late December with this new focus. With some ideas for inspiration (such as the web goat application) I will continue to research and refine the project. I will be bouncing ideas against my supervisor periodically to gauge how realistic the concepts are to incorporate.

Student Signature	
--------------------------	--

Month: December

What?

Finalized and submitted the project proposal. Felt a sense of accomplishment but also the pressure of meeting the deadline.

Experimented with interactive cybersecurity elements, drawing inspiration from WebGoat.

Began basic interface design, trying to visualize how users would interact with the content.

So What?

The submission marked a significant milestone and relieved some of the anxiety about the project's direction.

Initial feedback on the interface designs was encouraging, yet it highlighted my need to improve my design skills.

Struggled with balancing complexity and educational value in content.

Now What?

Plan to deepen my understanding of user-centered design.

Regular check-ins with my supervisor to ensure alignment with cybersecurity training objectives.

Start building the backend, focusing on the dual database integration.

Student Signature	
--------------------------	--

Month: January

What?

Began coding the backend, which was both challenging and rewarding.

Conducted initial usability tests, which were eye-opening in terms of user experience.

Integrated cybersecurity scenarios into the platform, which required a lot of creative thinking.

So What?

Realized the importance of robust backend architecture in supporting the frontend needs.

User feedback made me aware of the importance of intuitive design and clear navigation.

Felt excited and overwhelmed by the complexity of creating engaging content.

Now What?

Focus on simplifying the user interface based on feedback.

Continue enhancing my technical skills to tackle backend challenges.

Seek more feedback on content to ensure it is engaging and educational.

Student Signature	
--------------------------	--

Month: February

What?

Enhanced the gamification elements, which was fun but also technically demanding.

Improved database integration, learning a lot about performance optimization.

Expanded user testing, which brought diverse feedback and required quick adjustments.

So What?

Found joy in seeing users engage with the gamified elements.

Faced difficulties with database scalability, which tested my problem-solving skills.

User testing was stressful but ultimately very rewarding.

Now What?

Plan to further refine the gamification based on specific user feedback.

Address the scalability issues by exploring more advanced database solutions.

Increase user testing to refine features and improve user satisfaction.

Student Signature

Month: March

What?

Added feedback mechanisms and detailed progress tracking to the platform.

Conducted comprehensive testing, adjusting based on real user experiences.

Began preparing for the final project presentation, which was nerve-wracking.

So What?

Felt a deep connection to the project, seeing it nearly complete.

Stress and excitement as I began to finalize the project, knowing that presentation day was approaching.

Gained confidence in my ability to deliver a complex project.

Now What?

Focus on polishing every aspect of the project for the presentation.

Practice my presentation skills and prepare to answer difficult questions.

Ensure that all documentation is thorough and reflective of the work completed.

Student Signature

Month: April

What?

Presented the project, which was a culmination of months of hard work.

Received valuable feedback from the academic panel and potential stakeholders.

Reflected on the entire project journey, feeling proud and a bit exhausted.

So What?

Presentation day was exhilarating and a huge learning experience.

Feedback opened potential paths for further development or commercialization.

Felt proud of what I achieved and grateful for the learning journey.

Now What?

Explore potential enhancements and real-world applications for the project.

Consider further studies or career paths inspired by this project.

Document and share my experiences to help future students.

Student Signature

Student Name	Padraig McCauley
Student Number	20123744
Course	BSCCYBE4
Supervisor	Shivani Jaswal

Month: May

What?

Reflect on what has happened in your project this month?

This month I focused on getting my midpoint presentation together. I finalised my requirements (to the best of my ability as the time) and got my codebase to a presentable condition.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

The functionality still feels very basic in its implementation, but I am happy to have a final 'form' on the project. The UI and UX is quite clunky but having a more rounded functionality base mean I can start putting more thought into things like colour schemes and branding.

Now What?

What can you do to address outstanding challenges?

I feel I can round out the link between the email DB and the MSSQL DB so I will put things in place to make the backend functionality more seamless.

Student Signature	

Month: June

What? This month I finalised the link between the mongo DB and the MSSQL database and cleaned up the Ui of the email portal to make it more appealing to use.	
So What? I feel the look and feel of the email portal is the central draw of the application so I am happy to have it looking and feeling like an authentic email portal. The buttons/styling of the messages etc are as I had envisioned originally (styled off Gmail). The importing of the content is more rounded out now also.	
Now What? I hope to get my deployment on a cloud server next so I can put a CICD pipeline in place and implement the security measures needed to protect the server.	
Student Signature	

Month: July

What? This month I migrated my functionality to an AWS instance and implemented a CICD pipeline using Github and CircleCI. Security points were implemented to sure up access control etc and the app was moved to using https. Final styling points etc were done and my paperwork was completed.	
So What? The effort involved to get the project over the line has taught be a lot about time management and I feel my agile type approach has kept the workload focused and manageable.	

Now What?

This marks the end of my project and the end of my time in NCI. I am happy with the results and feel I have met the brief whilst also learning a lot about web development, security components of web application deployment and development and ultimately project management.

Student Signature	
--------------------------	--

Signature