OST
Ostschweizer
Fachhochschule

Securaware: Plattformlösung zum individualisierten Cybersecurity-Training für Privatpersonen

Studiengang Informatik
OST - Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Bachelorarbeit Frühjahrssemester 2025

Betreuer	Prof. Dr. Andreas Peter
Experte	M.A. Lukas Wunderlin
Gegenleser	Prof. Dr. Olaf Zimmermann
Autoren	Patrick Scheidegger, Mischa Binder

Danksagung

Wir möchten uns bei allen bedanken, welche uns während unserer Bachelorarbeit in jeglicher Form unterstützt haben.

Unser besonderer Dank gilt unserem Betreuer Prof. Dr. Andreas Peter, welcher uns über die gesamte Projektzeit hinweg unterstützt, betreut und inspiriert hat.

Zusätzlich bedanken wir uns bei Lukas Wunderlin für seine Vorschläge und sein Feedback beim Erstkontakttreffen.

Weiter möchten wir uns bei Prof. Dr. Olaf Zimmermann für sein Feedback und seine Inputs vom Zwischenmeeting bedanken. Zusätzlich möchten wir uns für das Feedback bezüglich der Softwarearchitektur bedanken.

Darüber hinaus bedanken wir uns bei Clemens Meier, ein wissenschaftlicher Mitarbeiter der OST, welcher den frontendseitigen Softwarecode analysiert und uns Feedback dazu gegeben hat.

Darüber hinaus möchten wir uns bei Alice Glaus für ihre Teilnahme am Usability-Test bedanken.

Abstract

Phishing zählt zu den am weitesten verbreiteten Angriffsformen im Bereich der Cybersecurity und gewinnt durch die voranschreitende Digitalisierung zunehmend an Relevanz im Alltag. Insbesondere Privatpersonen sind häufig unzureichend gegen derartige Angriffe geschützt und benötigen gezielte Massnahmen zur Steigerung ihrer Sicherheitskompetenz.

Im Rahmen dieser Bachelorarbeit wird ebendiese Lücke durch die Konzipierung einer Schulungsplattform und Entwicklung eines Prototypen geschlossen. Dieser Prototyp sensibilisiert Privatpersonen für Phishing mithilfe von theoretischen und praxisorientierten Inhalten, Simulationen und Beispielen. Bestehende Awareness-Lösungen, typische Vorgehensweisen von Angreifern und erkennbare Merkmale von Phishing-E-Mails werden in einem ersten Schritt analysiert, um daraus Anforderungen abzuleiten. Darauf aufbauend werden Features und eine Software-Architektur hergeleitet und durch Anwendung von state-of-the-art Software-Engineering-Praktiken der Prototyp implementiert. Mittels eines Usability-Tests wird die Benutzererfahrung evaluiert und optimiert.

Es wurde eine innovative und benutzerorientierte Schulungsplattform konzipiert und umgesetzt, welche Benutzer anhand von Phishing-Simulationen, Online-Kursen und Tests über Betrugsversuche im Internet präventiv informiert. Zukünftig kann Securaware iterativ weiterentwickelt und kommerziell verwendet werden.

Management Summary

Ausgangslage

Phishing ist eine Form des digitalen Betrugs, bei der Angreifer versuchen, über E-Mails, Webseiten oder andere digitale Kommunikationsmittel an vertrauliche Informationen von Nutzerinnen und Nutzern zu gelangen. Ziel ist es meist, persönliche Daten wie Passwörter oder Kreditkarteninformationen zu stehlen und für betrügerische Transaktionen zu verwenden - beispielsweise im Online-Banking oder in Online-Shops. Die Folgen sind finanzielle Schäden und ein zunehmendes Misstrauen gegenüber digitalen Diensten. Zwar existieren bereits Schulungsplattformen, diese richten sich jedoch primär an Unternehmen oder sind kostenpflichtig. Privatpersonen bleiben oft ohne geeignete, niederschwellige Unterstützung. Genau hier setzt diese Arbeit an.

Methodik / Vorgehen

In einem ersten Schritt werden die Anforderungen von Personen ohne Vorkenntnisse im Bereich Phishing analysiert. Parallel dazu erfolgt eine Marktanalyse bestehender Awareness-Lösungen. Auf Basis dieser Erkenntnisse wird ein technisches Konzept entwickelt und eine Webplattform mit dem Namen Securaware umgesetzt. Diese kombiniert Grundlagenwissen mit theoretischen und interaktiven Lerneinheiten sowie Phishing-Simulationen. Die Qualität der Implementierung wird durch automatisierte Tests sichergestellt. Die Benutzerfreundlichkeit der Plattform wird im Rahmen eines User-Testings mit vordefinierten Szenarien überprüft.

Ergebnisse

Im Rahmen des Projekts entsteht ein funktionaler Prototyp der Awareness-Plattform Securaware, der es unerfahrenen Nutzer:innen ermöglicht, sich interaktiv mit dem Thema Phishing auseinanderzusetzen. Die Plattform vermittelt Grundlagenwissen zu Angriffsmethoden, zeigt typische Erkennungsmerkmale von Phishing-Nachrichten auf und bietet relitätsnahe Trainingsszenarien. Das durchgeführte User-Testing zeigt, dass die Anwendung in weiten Teilen verständlich und intuitiv bedienbar ist. Die technische Architektur ist so ausgelegt, dass künftige Erweiterungen problemlos möglich sind.

Wirtschaftlicher Nutzen

Securaware bietet eine kostenlose, niederschwellige Möglichkeit für Privatpersonen, sich aktiv gegen Phishing-Angriffe zu wappnen. Die Plattform hilft dabei, das Risiko für Datenschutzverletzungen und daraus entstehende finanzielle Schäden zu reduzieren. Indem das Verständnis für digitale Bedrohungen gestärkt wird, leistet Securaware zudem einen Beitrag zur allgemeinen digitalen Kompetenz in der Bevölkerung.

Ausblick

In einer späteren Ausbaustufe könnten weitere, bereits konzeptionell erarbeitete, Funktionen wie ein „Phishing-Mentor“-Modul integriert werden. Damit wäre es möglich, dass erfahrene Nutzer:innen individuell zugeschnittene Trainingsszenarien für andere - etwa Familienmitglieder oder Bekannte - erstellen. Auch eine Erweiterung um weitere Gamification-Elemente oder zielgruppenspezifische Lernpfade, etwa für Jugendliche oder Senior:innen, ist denkbar.

Inhaltsverzeichnis

Glossar V
Abbildungsverzeichnis VII
Tabellenverzeichnis VIII
Codeverzeichnis IX
1. Einleitung 1
1.1. Ausgangslage 1
1.2. Aufgabenstellung 1
1.3. Rahmenbedingungen 1
2. Literaturrecherche und Marktanalyse 2
2.1. Phishing 2
2.1.1. Medium 2
2.1.2. Benutzerumgang mit Phishing 2
2.1.3. Technische Umsetzung 4
2.1.4. Gegenmassnahmen 5
2.2. Marktanalyse 7
2.3. Forschungsfrage 8
3. Methodik 9
4. Design 10
4.1. Anforderungen 10
4.1.1. Stakeholder & Personas 10
4.1.2. Funktionale Anforderungen 11
4.1.3. Nichtfunktionale Anforderungen 14
4.2. Produkt 16
4.2.1. Online-Kurs 16
4.2.2. Refresher 18
4.2.3. Erinnerung 18
4.2.4. Kurs-Test 18
4.2.5. Phishing-Mail-Simulation 18
4.2.6. WhatsApp Community 18
4.2.7. Phishing Mentor 19
4.3. Technisch 19
4.3.1. Software-Architektur 19
4.3.2. Tech-Stack 26
5. Technische Umsetzung 29
5.1. HTTP-Schnittstelle 29
5.1.1. OpenAPI 29
5.1.2. RESTful HTTP 29
5.1.3. HTTP-Error-Handling 30
5.2. Phishing Educator 30
5.2.1. Externe Softwarebibliotheken 31
5.2.2. Client-side Routing 32
5.2.3. Hey API 33
5.3. Phishing Backend 33

5.3.1. Funktionale Dekomposition ... 33
5.3.2. Ordnerstruktur ... 37
5.3.3. Externe Softwarebibliotheken ... 38
5.3.4. Connectivity zum Phishing Forwarder ... 38
5.3.5. HTTP-Model-Generierung ... 39
5.3.6. Implementierung der Online-Kurs-Tests ... 39
5.4. Phishing Forwarder ... 40
5.5. Securaware Proxy ... 40
5.5.1. Statische Konfigurationen ... 42
5.5.2. Dynamischen Konfigurationen ... 42
5.6. Phishing Mail Storage ... 44
5.6.1. Entitäten ... 44
5.6.2. Schemamigration ... 46
5.7. Docker ... 47
5.7.1. Container-Orchestrierung ... 47
5.7.2. Netzwerk ... 47
5.7.3. Phishing Backend ... 48
5.7.4. Phishing Educator ... 50
5.8. Softwarecode-Testkonzept ... 51
5.9. Usability-Test ... 53
5.9.1. Testaufbau und Methodik ... 53
5.9.2. Durchführung ... 53
5.9.3. Auswertung und Erkenntnisse ... 54
5.9.4. Fazit ... 54
5.10. Betriebskonzept ... 54
5.11. Continuous Integration und Continuous Delivery (CI/CD) ... 55
5.11.1. Test-Stage ... 56
5.11.2. Publish und Deploy-Stage ... 58
5.12. Betriebskonzept nach der Bachelorarbeit ... 59
6. Diskussion und Schlussfolgerung ... 60
6.1. Resultate ... 60
6.2. Diskussion ... 60
6.3. Limitationen & Ausblick ... 61
7. Bibliographie ... 63

Glossar

***ACME* – Automatic Certificate Management Environment**: Protokoll, welches das Aufsetzen von digitalen Zertifikaten über einer Zertifizierungsstelle automatisiert. 42

***API* – Application Programming Interface**: Programmier- oder Anwendungsschnittstelle über welche eine Software oder ein System angesprochen werden kann. 19

***BDD* – Behavior-Driven Development**: Ansatz um unter anderem automatisierte Testfälle gemäss dem Given-When-Then-Prinzip zu strukturieren. 52

***CSS* – Cascading Style Sheets**: Stylesheet-Sprache, welche das Design von Webseiten definiert. 26, 31

***DNS* – Domain Name System**: Protokoll, welches unter anderem Domains und IPv4-Adressen miteinander verbindet. 5

***DOM* – Document Object Model**: Schnittstelle, mit welcher auf Hypertext Markup Language (HTML) -Elemente zugegriffen werden kann. 31

***ESNI* – Encrypted Server Name Indication**: Erweiterung des Transport Layer Security (TLS) Standards, mit welcher unterschiedliche Zertifikate basierend auf der Domain zurückgegeben werden können. VIII, 22

***HTML* – Hypertext Markup Language**: Textbasierte Auszeichnungssprache, welche den Inhalt und die Struktur von Webseiten definiert. V, 26, 31

***HTTP* – Hypertext Transfer Protocol**: Open Systems Interconnection (OSI) Schicht 7 (Anwendung) Protokoll, das zur Übetragung von Daten und Webseiten verwendet wird. 34, 42

***IDL* – Interface Definition Language**: Sprache zur Beschreibung von Schnittstellen 29

***JVM* – Java Virtual Machine**: Laufzeitumgebung für Java-Bytecode basierte Programme. 27

***JWT* – JSON Web Token**: Token, welcher Behauptungen (eng. claims) im JSON-Format enhält, die für die Authorisierung verwendet werden. 38

***MTA* – Mail Transfer Agent**: Server, welcher E-Mails entgegennimmt und über Simple Mail Transfer Protocol (SMTP) weiterversendet. 40

***ORM* – Object-relational mapping**: Software, welche programmiersprachspezifische Primitiven in eine relationale Datenbank abbilden kann. VIII, 22, 37, 38, 47

***RUP* – Rational Unified Process**: Rational Unified Process (RUP) ist ein iteratives und phasenbasiertes Softwareentwicklungsmodell. Es unterteilt den Entwicklungsprozess in vier Phasen: Inception, Elaboration, Construction und Transition.

***SDK* – Software Development Kit**: Eine Sammlung von Tools, Funktionen, Bibliotheken und Dokumentationen, die Entwickler benötigen, um Anwendungen für eine bestimmte Plattform oder ein bestimmtes System zu erstellen. 31

***SMTP* – Simple Mail Transfer Protocol**: Open Systems Interconnection (OSI) Schicht 7 (Anwendung) Protokoll, das zum Austausch von E-Mails dient. V

***SSG* – Static Site Generation**: Webentwicklungsansatz, bei dem die Webapplikation zur build time kompiliert wird, sodass die Artifakte statisch sind und sich nicht mehr verändern. 21

TLS – **Transport Layer Security**: Open Systems Interconnection (OSI) Protokoll, das zum sicheren Transfer von Daten verwendet wird. V, 42, 50

UML – **Unified Modeling Language**: Grafische Modellierungssprache, um Informatik-spezifische Anliegen visualisieren zu können. 11

UX – **User Experience**: Wie ein Nutzer eine Software erlebt und darauf reagiert - von der ersten Nutzung bis zum abschliessenden Eindruck. 15, 54

Abbildungsverzeichnis

Abbildung 1 FireFox Suchleiste mit hervorgehobener Domain 6
Abbildung 2 Use Case Diagramm 12
Abbildung 3 Modulthematiken und -Inhalte dee Online-Kurse 16
Abbildung 4 C4 Context-Diagramm von Securaware 20
Abbildung 5 C4 Container-Diagramm von Securaware 21
Abbildung 6 Beschreibung der REST Maturity Levels. Übernommen aus [1] 30
Abbildung 7 Beispiel einer Onion Architecture. Übernommen aus [2] 34
Abbildung 8 Beispiel einer Clean Architecture. Übernommen aus [3] 35
Abbildung 9 Beispiel einer Hexagonal Architecture. Übernommen aus [4] 35
Abbildung 10 Beispiel einer Layered Architecture. Übernommen und angepasst nach [3] . 36
Abbildung 11 Beispiel einer Vertical Slice Architecture. Übernommen aus [5] 36
Abbildung 12 Ordnerstruktur gemäss Onion Architecture des Phishing Backends 37
Abbildung 13 Simplifizierte Traefik-Konfigurationen im Überblick. Eigene Darstellung unter Verwendung des Traefiklogos von [6] 41
Abbildung 14 Entity-Relationship Diagram von Securaware 45
Abbildung 15 Testingpyramide. Die verschiedenen Arten von automatisierten Testfällen werden in Form einer Pyramide aufgelistet. 51
Abbildung 16 Betriebskonzept von Securaware in Form eines Deploymentdiagramms 55
Abbildung 17 Continuous Integration und Continuous Delivery Main-Build-Pipeline 55

Tabellenverzeichnis

Tabelle 1 NFR1 Performance 14
Tabelle 2 NFR2 Security 15
Tabelle 3 NFR3 Usability 15
Tabelle 4 NFR4 Portability 15
Tabelle 5 NFR5 Accessibility 15
Tabelle 6 Phishing Educator y-Template about performance 21
Tabelle 7 Securaware Proxy y-Template about IPv4 addresses 22
Tabelle 8 Securaware Proxy y-Template about Encrypted Server Name Indication (ESNI) 22
Tabelle 9 Phishing Backend y-Template about programming language 22
Tabelle 10 Phishing Backend y-Template about the usage of an Object-relational mapping (ORM) 22
Tabelle 11 Phishing Backend y-Template about folder structure 23
Tabelle 12 Phishing Backend y-Template about using a web framework 23
Tabelle 13 Phishing Backend y-Template about how to describe the API 23
Tabelle 14 Phishing Backend y-Template about generating code out of the API 24
Tabelle 15 Phishing Backend y-Template about error handling 24
Tabelle 16 Phishing Backend y-Template about error handling 24
Tabelle 17 Phishing Backend y-Template about monitoring the runtime behaviour 24
Tabelle 18 Phishing Forwarder y-Template about risk management 25
Tabelle 19 Phishing Mail Storage y-Template about SQL vs NoSQL 25
Tabelle 20 Phishing Mail Storage y-Template about PostgreSQL 25
Tabelle 21 Phishing Mail Storage y-Template about normalization forms 26
Tabelle 22 Vergleich JavaScript-Frameworks und Libraries. Bewertung [0-4], wobei 0 die tiefste und 4 die höchste Bewertung ist. 27
Tabelle 23 Vergleich Programmiersprachen für das Phishing Backend. 27
Tabelle 24 Externe Softwarebibliotheken des Phishing Educators 31
Tabelle 25 Externe Softwarebibliotheken des Phishing Backends 38
Tabelle 26 Beschreibung der verschiedenen Arten von Testfällen 52
Tabelle 27 Dikussion, Vergleich der Anforderungen und Resultate 61

Codeverzeichnis

Code 1 Beispiel-RFC-9457-Fehlermeldung 30
Code 2 HTTP-Client-Code-Generierung mit Hey API 33
Code 3 Simplifizierte Go-HTTP-Server-Code-Generierung 39
Code 4 Berechnung der Anzahl falschen oder fehlenden Antworten mit der Kontravalenz 40
Code 5 Simplifizierte statische Traefik-Konfigurationen (traefik.yaml) 42
Code 6 Simplifizierte dynamische Traefik-Konfigurationen (traefik_dynamic.yaml) 43
Code 7 Simplifiziertes und auf Docker-Netzwerk reduziertes compose.yaml 48
Code 8 Dockerfile des Phishing Backend 49
Code 9 Simplifiziertes Dockerfile des Phishing Educators 50
Code 10 GitLab-Jobs für automatisierte Testfälle 57
Code 11 GitLab-Jobs für continuous delivery 58

1. Einleitung
Nachfolgend wird die Ausgangslage dieser Bachelorarbeit beschrieben, die Aufgabenstellung erläutert und die Rahmenbedingungen definiert. Diese Informationen basieren auf dem Arbeitsauftrag, welcher im Anhang erhältlich ist.

1.1. Ausgangslage
Cyberangriffe treten zunehmend häufiger auf. Nicht-Experten sind am stärksten von diesen Angriffen betroffen. Unternehmungen schulen ihre Mitarbeiter bereits über diverse Schulungsplattformen. Privatpersonen kommen zu kurz, für diese gibt es wenige vergleichbare Schulungsplattformen geschweige denn kostenlos. Als Produkt dieser Arbeit wird ein Prototyp erwartet, welcher Privatpersonen interaktiv zum Thema Cybersecurity schult - nämlich „Securaware“.

Die Idee und der Name des Produkts wurde von den Autoren konzipiert. Als Vorarbeit wurde bereits eine Marktanalyse für bestehende Awareness-Programme, die sich hauptsächlich auf Angebote für Unternehmen spezialisiert haben, erhoben.

1.2. Aufgabenstellung
Es soll eine Online-Schulungsplattform zum Thema Phishing erstellt werden. Hierfür sollen aktuell auf dem Markt existierende Cybersecurity-Schulungsplattformen evaluiert und analysiert werden. Anforderungen und Features für eine neue Schulungsplattform für Privatpersonen sollen anhand dessen abgeleitet werden. Der aktuelle Stand der Wissenschaft bezüglich Cybersecurity soll erhoben werden und in die Applikation miteinfliessen. Eine technische Architektur gemäss Best Practices, Industriestandards und vermitteltem Wissen der OST soll entworfen und umgesetzt werden. Die Arbeit soll in einem technischen Bericht dokumentiert werden. Dieser soll den Bachelorstandards folgen, unter anderem bezüglich Struktur, Literaturrecherche, Verweise und Qualität. Zusätzlich sollen OST-spezifische Anforderungen erfüllt und Dokumente erstellt werden. Dazu gehören unter anderem die Bachelorprüfung, die Zwischenpräsentation, das Poster und das Broschürenabstract.

1.3. Rahmenbedingungen
Die OST erkennt diese Bachelorarbeit mit 12 ECTS-Credits pro Person an. Pro Credit werden 30 Stunden Aufwand erwartet. Entsprechend werden 360 Stunden Aufwand pro Person oder 720 Stunden insgesamt erwartet.

2. Literaturrecherche und Marktanalyse

In diesem Kapitel wird mithilfe einer Literaturrecherche ein Common Ground zum Thema Phishing geschaffen und aufgezeigt, wie Phishing praktiziert wird und auf welche Arten man sich und andere davor schützen kann. Anschliessend werden in einer Marktanalyse bestehende Lösungen analysiert. Aus diesen Informationen wird abschliessend die Forschungsfrage dieser Arbeit formuliert.

2.1. Phishing

Was ist mit Phishing gemeint? Unter Phishing versteht man, den Versuch eines Angreifers, sensible Daten eines Opfers zu erlangt. Solche Daten sind insofern wertvoll, da der Angreifer mit diesen eine falsche Identität vortäuschen kann, sei es in einem Login durch ein Passwort oder durch anderweit sensible Daten, die nur das Opfer wissen dürfte. Zusätzlich können diese sensiblen Daten natürlich auch weiterverkauft werden. Diese sensiblen Daten werden oft mit Betrug erlangt.

2.1.1. Medium

Diese generelle Definition von Phishing erlaubt Angreifern diverse Medien zu verwenden:

- E-Mail
- SMS
- Telefonanrufe
- Social Media
- Blogs und Foren
- Quick Response Code Login Jacking (QRLJacking): Hier werden Applikationen, welche einen QR-Code fürs login verwenden, angegriffen.
- Physische Post
- Gespräche in Person

E-Mail ist hierbei aufgrund des geringen Aufwands und der riesigen Verbreitung der Technologie das meistverwendete Medium.

2.1.2. Benutzerumgang mit Phishing

Angreifer versuchen sensible Daten des Opfers zu klauen. Entsprechend wird der Benutzer und nicht das Technische Komponente angegriffen. Aus diesem Grund ist es von hoher Bedeutung zu verstehen, wie ein Benutzer Entscheidungen trifft und welche Aspekete diese Entscheidungen beeinflussen. Dieses Unterkapitel basiert auf den Ergebnissen und Erkenntnissen von [7].

2.1.2.1. Beeinflussende Aspekte

Gemäss [7] gibt es drei Aspekete, welche die Entscheidungen eines Benutzers hinsichtlich eines Phishing-Angriffes prägen:

- **Externe Informationen**: Externe Informationen sind Informationen, welche über die Benutzerschnittstelle angezeigt werden. Im Phishing-Kontext wäre dies die betrügerische E-Mail, SMS, Nachricht oder weiteres.

- **Wissen und Kontext**: Das Verständnis, das die Benutzerin und der Benutzer von der Welt, der Technik und Betrug besitzen.
- **Erwartungen**: Aufgrund der zwei genannten Aspekte entstehen Erwartungen.

2.1.2.2. Aufbau des Entscheidungsprozess

Gemäss [7] ist der Entscheidungsprozess in drei aufeinanderfolgenden Phasen unterteilt:
1. das Wahrnehmen der Situation
2. das Ausdenken von möglichen Reaktionen
3. das Erstellen von Bewertungskriterien und der Auswahl einer Massnahme

Zur Wahrnehmung der Situation gehören folgende drei Komponenten:
- Teilnehmer
  - Absender, Auslöser
  - Der Profitierende
  - Subjekt, sensible Daten
  - Empfänger
- Kausalität
  - Grund der Interkation
  - Ziel und Zweck der Interkation
- Empfehlung, Aufforderung
  - Explizite, suggerierte Aktion, um zum Ziel zu kommen
  - Implizite Aktion

Ein betrügerischer Angriff erscheint glaubwürdig, wenn die Komponenten zueinander kohärent sind. Ein Beispiel für einen unglaubwürdigen Angriff ist eine Phishing-Email von der angeblichen Bank des Benutzers, welchen diesen um seine Logindaten bittet, damit sie sein Konto entsperren können. Dies ergibt keinen Sinn, da die Bank als Autorität den vollen Zugriff auf das Konto des Benutzers hat und nicht auf seine Logindaten angewiesen ist. Die Kausalität (das Entsperren des Kontos) und die Teilnehmer (die Bank als Absenderin und gleichzeitig Zuständige) sind entsprechend im Konflikt zueinander. Ein weiteres Beispiel für einen unglaubwürdigen Angriff ist eine Phishing-Email eines angeblichen Lieferdienstes, welches ein Paket an den Benutzer zustellen möchte, es jedoch aufgrund offener Lieferkosten nicht vollziehen kann. Entsprechend soll der Benutzer die Zugangsdaten zu seinem E-Banking-Account preisgeben, damit der Lieferdienst die Kosten negieren kann. In diesem Beispiel stehen die Kausalität und die Empfehlung im Konflikt: Es ergibt keinen Sinn, dass der Lieferdienst die offenen Kosten selber mit den Bankdaten des Kunden decken möchte. Es wäre viel plausibler, dass der Lieferdienst dem Kunden eine Rechnung ausstellt.

Die Benutzerinteraktion **beim Ausdenken von möglichen Reaktionen** ist gering, die meisten Phishing-Angriffe bieten dem Benutzer bereits eine angebliche Lösung an. Hinter der Lösung versteckt sich aber ein Betrug, sei es eine gefälschter Link oder eine gefälschte Organisation. Es gibt auch entsprechende Angriffe ohne explizite Lösungsvorschläge. Hierbei werden implizite Lösungsvorschläge als Angriff verwendet oder es wird auf falsche Lösungen des Benutzers gepocht. Ein Angriff mit einem impliziten Lösungsvorschlag kann

wie folgt aussehen: Eine E-Mail weist zwei Links auf, der erste dient als Hauptlink und der zweite als Backup, falls der Erste nicht funktioniert. Der erste Link zeigt auf die korrekte Domain der Organisation, aber wurde absichtlich so gebaut, dass die Organisation selbst nichts anzeigt. Zum Beispiel zeigt der Link auf ein nicht existierendes Buch in einer Bibliothek. Die richtige Webseite der Bibliothek wird geladen, aber diese kann keine Informationen zum Buch anzeigen, da das Buch nicht existiert. Der zweite Link hingegen zeigt auf eine fälschliche Domain, welche vom Angreifer geführt wird. Der implizite Vorschlag besteht nun darin, dass der Benutzer doch den zweiten Link anklicken soll und dieser legitim ist. Weil der erste Link legitim ist, sinkt das Sicherheitsverständnis des Benutzers, weshalb dieser den zweiten Link mit weniger Vorsicht anklicken wird.

Die Entscheidung, ob die vorgeschlagene Handlung ausgeführt wird, steht **beim Erstellen von Bewertungskriterien und der Auswahl einer Massnahme** im Mittelpunkt. Die Kriterien werden vom Benutzer erstellt, um Gewinne und Risiken zu evaluieren. Diese Kriterien bestehen entweder bereits oder werden vom Benutzer erstellt. Sie sind von sehr vielen Aspekten abhängig, unter anderem vom Wissen, der Erfahrung, den Präferenzen und dem Zustand des Benutzers. Die Entscheidungsgrundlage des Benutzers anzugreiffen ist entsprechend komplex und schwierig. Gemäss [7] kann man sich jedoch auf Gemeinsamkeiten beschränken und diesen Aspekt vereinfachen: „However, most phishing attacks analysed didn't take advantage of the differences between users, instead they take advantage of what users have in common" [7]. Solche Gemeinsamkeiten können zum Beispiel sein, dass man Aufforderungen von Autoritäten weniger bis hin zu garnicht hinterfragt. Ein Angreifer könnte sich entsprechend als den Staat des Niederlassungslandes des Opfers ausgeben um so an Vertrauen zu gelangen. Ein weiteres Beispiel ist, dass Benutzer ein sicheres Bankkonto haben möchten. Ein Angreifer könnte sich als Bank des Opfers ausgeben und einen Link anbieten, der angeblich mit nur einem Klick die Sicherheit des Kontos des Benutzers verschärft.

2.1.3. Technische Umsetzung

In diesem Unterkapitel werden die technischen Aspekte, wie ein Phishing-Angriff stattfindet, beschrieben.

2.1.3.1. URL spoofing

Viele Angriffe setzen auf gefälschte Links und URLs. Dieses Unterkapitel listet auf, was für Fälschungen es gibt und wie diese funktionieren. Die Bezeichnungen für diese Fälschungen werden aus [8] übernommen.

Bad domain name: Die richtige Domain wird mit einer sehr ähnlichen Domain verfälscht. Beispiel:
richtige URL: www.ba**n**k-of-switzerland.ch
gefälschte URL: www.ba**m**k-of-switzerland.ch

Shortened URLs: Wie bei bad domain name, wird auch hier eine Domain verwendet, welche der richtigen Domain stark ähnelt. Diese URL wird jedoch hinter eine verkürzte URL versteckt, sodass der Benutzer nicht direkt die falsche URL sieht.

Beispiel: Beim Öffnen von bit.ly/abcdef wird der Benuter zu www.ba**m**k-of-switzerland.ch/this-is-a-very-very-very-long-url anstatt www.ba**n**k-… weitergeleitet.

Host name obfuscation Auch dieser Ansatz basiert auf bad domain name. Es wird eine Fälschung erstellt und anstelle des benutzerfreundlichen Domainnamens (Domain Name System (DNS) -Eintrag) wird die IPv4-Adresse im Link verwendet. Der Link zeigt die IPv4-Adresse an, populäre Browser hingegen zeigen die Domain.
Beispiel: www.ba**m**k-of-switzerland.ch (IPv4: 7.7.7.7). Gefälschter Link: https://7.7.7.7. Browseransicht: www.ba**m**k-of-switzerland.ch

Internationalized domain name (IDN) homograph attack: Dieser Ansatz basiert auf Homoglyphen. Homoglyphen sind sehr ähnlich oder gar gleich aussehende Schriftzeichen. Zum Beispiel sehen das lateinische, kleine o (Unicode: U+006F) und das kyrillische kleine o (Unicode: U+043e) für den Menschen identisch aus. Beispiel: www.google.com und www.google.com sehen identisch aus, verweisen aber auf zwei unterschiedliche Domains. Populäre Browser jedoch zeigen die Google-Domain bestehend aus den kyrillischen o wie folgt an: www.xn–ggle-55da.com.

2.1.4. Gegenmassnahmen

Dieses Unterkapitel listet Gegenmassnahmen gegen Phishing auf. Ziel ist es, den Benutzer für betrügerische E-Mails und Webseiten zu sensibilisieren. Dies kann über zwei Massnahmen bewerkstelligt werden; in dem der Benutzer geschult und auf die Risiken hingewiesen wird und über die Empfehlung, Tools zu installieren, welche betrügerische Inhalte erkennen und den Benutzer warnen.

2.1.4.1. Spam Filtering

Gemäss „Filtering and Detection of Real-Time Spam Mail Based on a Bayesian Approach in University Networks“ [9] gibt es einige Methoden und Technologien, um Phishing-E-Mails zu erkennen. Diese sind unter anderem:

- Blacklist und whitelist filtering: E-Mails von bestimmten IP-Adressen, URI oder weiteren Attributen werden entweder strikt akzeptiert (whitelist) oder abgelehnt (blacklist). Domain Name System-based blackhole list (DNSBL) ist ein Beispiel einer Blacklist von URIs.
- Keyword and phrase filtering: Bestimmte Schlüsselwörter oder -ausdrücke weisen auf Phishing hin.
- Künstliche Intelligenz: Vortrainierte Modelle, welche anhand von Testdaten und Statistiken E-Mails evaluieren können.
- Technische Standards und Protokolle: Diese werden genauer in Abschnitt 2.1.4.3 beschrieben.

2.1.4.2. Generell technische Gegenmassnahmen

Multi-factor authentication (MFA); two-factor authentication (2FA): MFA und 2FA verlangen zwei oder mehr unabhängige Faktoren zur Authentifizierung. Diese Faktoren sind unter anderem:

- „Wissen“, etwas, das der Benutzer weiss, wie zum Beispiel ein Passwort

- „Besitz“, etwas, das der Benutzer besitzt, wie zum Beispiel eine Bankkarte oder ein Smartphone
- „Inhärenz“, etwas, das der Benutzer ist, wie zum Beispiel ein Fingerabdruck
- „Ort“, ein Ort, an welchem sich der Benutzer gerade befindet

Betrügerische Inhalte versuchen, an sensible Daten des Opfers zu gelangen, also etwas, das der Benutzer weiss oder besitzt. Andere Faktoren sind resistent gegenüber Phishing-Angriffe. Durch die Aktivierung von MFA in sensiblen Services wie zum Beispiel E-Banking, E-Governence oder gesundheitsbezogenen Services (wie zum Beispiel ein Verwaltungstool von elektronischen Patientenakten) werden diese resistent gegenüber Phishing-Angriffe: Der Angreifer kann in seinem besten Fall an etwas gelangen, was der Benutzer weiss, aber nicht an die anderen Faktoren, wodurch der Angriff nicht erfolgreich durchgeführt werden kann.

Aktive und passive Browser-Toolbars: Phishing-Angriffe versuchen oftmals den Benutzer auf eine gefälschte oder gefährliche Webseite zu locken. Aktive und passive Browser-Toolbars schreiten zur Tat, wenn der Benutzer auf solch eine Webseite gelockt wird: Passive Toolbars zeigen dem Benutzer eine Warnung an. Aktive unterbrechen zusätzlich noch die Benutzerinteraktion, in dem der Benutzer die Warnung bestätigen muss. Aktive Toolbars sind effektiver als Passive: „Active toolbars […] are more effective.“ [10]. Die Toolbars verwenden intern URI DNSBL. Die Toolbars können Angriffe verhindern, werden aber mit fortschreitender Zeit weniger beachtet. Grund dafür ist „Banner Blindness“.

Browser-Domain-Highlighter: Diese Art von Tool ist gegen URL Spoofing ausgerichtet: Falls der Benutzer den betrügerischen Link anklickt, kann dieser an der Domain erkennen, dass der Link ihn oder sie getäuscht hat. Sinn und Zweck des Tools ist es die Domain hervorzuheben und all die anderen Teile abzuschwächen um den Fokus des Benutzers auf den relevanten Teil zu lenken. Der Browser Firefox ist hierfür ein Musterbeispiel, nativ unterstützt es dies bereits gemäss Abbildung 1.

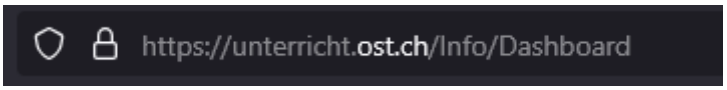


Abbildung 1: FireFox Suchleiste mit hervorgehobener Domain

2.1.4.3. Standards und Protokolle

Diese Standards und Protokolle stellen die Confidentiality oder Integrity gemäss CIA Triad sicher. Ein Beispiel für die Integrity ist die Authentifizierung des Absenders. Ein Beispiel für die Confidentiality ist die Verschlüsselung des E-Mail-Verkehres. Dies sind einige der Standards und Protokolle:

- Pretty Good Privacy (PGP): Über ein selbstgeneriertes public-private-key-Paar werden Confidentiality und Integrity durch Verschlüsselung und digitale Signaturen gewährleistet.
- Secure/Multipurpose Internet Mail Extensions (S/MIME): Ähnlich wie PGP mit dem grossen Unterschied, dass das public-private-key-Paar von einer Certificate Authority verifiziert werden muss.

- Sender Policy Framework (SPF): Verifiziert, ob der angegebene Absender das E-Mail versendet hat. Bei Erhalt der E-Mail wird anhand der Domain des Absenders dessen DNS-Server nach SPF-Einträgen abgefragt und mit der Sender-IP-Adresse der E-Mail verglichen. Wird keine Übereinstimmung gefunden, so wird angenommen, dass die E-Mail nicht vom angegebenen Absender stammt.
- DomainKeys Identified Mail (DKIM): Verifiziert, dass die E-Mail während des Transfers nicht verändert wurde. Hierfür wird mit einem public-private-key-Paar die E-Mail signiert. Der Empfänger überprüft die Signatur mit dem DNS-DKIM-Eintrag des Absenders, in welchem der Public-Key des Absenders zur Verfügung gestellt wird.
- Domain-based Message Authentication, Reporting and Conformance (DMARC): Basiert auf DKIM und SPF und erweitert diese um Reporting und Angaben, wie die E-Mail bei einem Fehlschlag behandelt werden soll.
- Brand Indicators for Message Identification (BIMI): Basiert auf DMARC und zeigt zusätzlich das Logo der Unternehmungen des Absenders.

2.2. Marktanalyse

Um die Notwendigkeit dieser Anwendung und die bereits auf dem Markt eingesetzten Methoden zur Sensibilisierung zu verstehen, wird eine Markt- und Bedarfsanalyse durchgeführt. Dafür wird durch Internet-Recherchen nach aktuellen Angeboten recherchiert. Dabei werden die Angebote der Anbieter phished.io, das Security Awareness Training der Swisscom und knowbe4.com verglichen.

Vergleich

Die Analyse zeigt auf, dass sich das Angebot der betrachteten Anbieter in sehr vielen Aspekten überschneidet. So richten sich sämtliche Angebote ausschliesslich an Unternehmen und setzen Gamifizierung ein, um die Zielgruppe dazu zu bewegen, die Schulungs-Kurse zu absolvieren. Auch liegt der Fokus bei sämtlichen auf dem Markt angebotenen Dienstleistung stark auf E-Mail Phishing. Andere Angriffsvektoren wie per Post, SMS oder Blogs und Foren werden dabei vernachlässigt.

Fazit

Daraus lässt sich ableiten, dass der Markt für Unternehmen in Bezug auf E-Mail Phishing sehr gut gesättigt ist. Personen, die jedoch nicht in einem Unternehmen tätig sind, das ein solches Security Awareness Training durchsetzt, oder im Berauf kaum bis gar nicht mit der Informatik konfrontiert werden, werden somit unzureichend trainiert und sind auf sich alleingestellt.

In den Recherchen konnte kein Angebot gefunden werden, welches sich explizit an Privatpersonen richtet. Es kann daher angenommen werden, dass aktuell kein solches Angebot auf dem Markt existiert. Der aus dieser Arbeit resultierende Prototyp, resp. das Produkt, hat als Ziel diese Lücke zu schliessen und das Angebot somit auf individuelle Personen auszuweiten. Dies trägt der Sicherheit der Gesellschaft im Umgang mit der IT und dem Internet bei.

2.3. Forschungsfrage

Die vorangegangene Literaturrecherche zeigt, dass Phishing eine der häufigsten und gefährlichsten Bedrohungen im digitalen Raum darstellt. Besonders Privatpersonen sind häufig Ziel solcher Angriffe, verfügen jedoch nicht über das notwendige Wissen und die Ressourcen, um sich wirksam dagegen zu schützen. Die Analyse bestehender Phishing-Awareness-Plattformen zeigt auf, dass sich viele Angebote vorrangig an Unternehmen und deren Mitarbeiter richten. Plattformen speziell für Privatnutzer sind seltener, häufig weniger interaktiv und bieten nur begrenzte Möglichkeiten zur praxisnahen Anwendung des Gelernten.

Daraus ergibt sich der Bedarf, ein Konzept für eine Awareness-Plattform zu entwickeln, das sich gezielt an Privatpersonen richtet und diesen sowohl theoretisches Wissen als auch praktische Fähigkeiten zur Erkennung und Vermeidung von Phishing-Versuchen vermittelt. Ziel soll es sein, den Lerneffekt durch geeignete didaktische Methoden wie interaktive Lernmodule, realitätsnahe Beispiele oder simulierte Phishing-Angriffe zu maximieren.

Auf Grundlage dieser Erkenntnisse ergibt sich folgende zentrale Forschungsfrage:

„Wie könnte eine digitale Plattform gestaltet und umgesetzt werden, die Privatpersonen bestmöglich im Bereich Phishing aufklärt und durch theoretische Inhalte sowie praxisnahe Beispiele zur Prävention befähigen kann?“

3. Methodik

Zur Beantwortung der Forschungsfrage werden in einem ersten Schritt Zielgruppen definiert, welche mit dem Produkt angesprochen werden sollen, damit die bestehende Marktlücke im Security-Awareness-Bereich, bezogen auf Phishing, geschlossen werden kann.

Aus dem daraus resultierenden Ergebnis, sowie jenem der Marktanalyse, werden funktionale Anforderungen in Form von Use Cases definiert, welche die Bedürfnisse der Zielgruppe bestmöglichst abdecken. Weiter werden nicht funktionale Anforderungen an die Qualität und zu liefernde Benutzererfahrung des Produkts festgelegt.

Im Anschluss wird eine Architektur ausgearbeitet, die den Anforderungen bestmöglich gerecht werden kann. Weitere Aspekte wie die Erfahrung oder Popularität werden bei der Wahl der Architektur zur Entscheidungsfindung hinzugezogen.

In einem nächsten Schritt wird aufgrunddessen ein Prototyp entwickelt. Dafür werden verschiedenste Entscheidungen wie die Wahl der Programmiersprache oder der Einbindung eines Frameworks getroffen. Anschliessend wird mithilfe eines Durchstichs verifiziert, dass das gewählte Setup den Anforderungen der geplanten Applikation gerecht werden kann. Darauf folgt die Einrichtung eines Test-Frameworks, welches zur Erstellung von automatisierten Testfällen herbeigezogen werden kann, um den zu implementierenden Code auf seine Korrektheit überprüfen zu können. Weiter werden in iterativen Entwicklungszyklen die zuvor erarbeiteten funktionalen Anforderungen implementiert. Parallel dazu wird ein automatisiertes Deployment-Konzept erarbeitet, womit die Applikation automatisch geprüft und bereitgestellt werden kann. Dadurch steigert sich die Effizienz während der Entwicklungsphase durch kürzere Zyklen enorm.

Mithilfe eines Code-Reviews als auch einem Usability-Test können Peer-Reviewer und potenzielle Benutzer ihr Feedback zur erarbeiteten Lösung abgeben, welches wiederum in die Entwicklung miteinbezogen werden kann.

4. Design

Dieses Kapitel befasst sich mit den Anforderungen an das Produkt, sowie dessen Gestaltung und Konzeption. Zur Gestaltung und Konzeption gehören sowohl die effektiven Features, welche die Applikation anbieten soll, als auch die technischen Aspekte.

4.1. Anforderungen

Dieses Unterkapitel definiert potenzielle Benutzer und leitet aufgrund dieser und der Marktanalyse funktionale und nichtfunktionale Anforderungen ab.

4.1.1. Stakeholder & Personas

Um die Zielgruppen und deren Anforderungen zu verstehen werden nachfolgend 4 Personas erstellt. Es handelt sich dabei um rein fiktive Personen, welche je eine Zielgruppe und ihre Anforderungen repräsentieren.

4.1.1.1. Junge Erwachsene

Der 21-jährige Medizinstudent Lukas Schneider fühlt sich unsicher im Umgang mit den technischen Mitteln. Durch seine Gutherzigkeit entwickelt er schnell Vertrauen zu neuen Bekanntschaften. Damit wird er zu einem leichten Ziel für potenzielle Angreifer.

Aus diesem Grund möchte sich Lukas vor Betrug im Internet schützen, indem er sich über aktuelle Angriffe und die angewendeten Methoden informiert. Gleichzeitig möchte er sich gut auf seine berufliche Laufbahn vorbereiten, da er täglich im Umgang mit besonders schützenswerten Personendaten stehen wird.

4.1.1.2. Phishing-unerfahrene Beschäftigte

Die 34-jährige Marianne Meier arbeitet in der Finanzbuchhaltung eines Klein-/Mittelgrossen Unternehmens. Sie fühlt sich im Umgang mit der Technik relativ sicher, ist sehr aufgeschlossen, modern und vorausschauend.

Ihr Unternehmen ist noch relativ klein, wodurch der Einsatz eines Security Awareness Trainings bisher nicht in Betracht gezogen wurde. Marianne arbeitet jedoch täglich mit vertraulichen Daten aus der Finanzbuchhaltung und ist die erste Ansprechsperson im Unternehmen, wenn es um Finanzen geht. Dadurch wird Marianne schnell zum möglichen Ziel für Betrüger im Internet. Desshalb möchte Marianne in der Lage sein, mögliche Angriffe auf sie bzw. ihr Arbeitgeber frühzeitig zu erkennen.

4.1.1.3. Rentner

Für den 69-jährigen Rentner, Michael Fischer, ist der Umgang mit der Technik noch sehr neu und er gewöhnt sich nur sehr langsam an die stetige Entwicklung. Durch seine Leichtgläubigkeit lässt er sich sehr leicht beeinflussen, was ihn zu einem leichten Ziel für Betrüger macht.

Durch diverse Berichte über Betrugsmaschen aus den Medien fühlt er sich sehr unsicher und er und seine Angehörigen haben Angst, erkönnte schnell zum Opfer werden. Um dies zu verhindern, möchte er in der Lage sein, Angriffe auf sich zu erkennen und im Falle eines Angriffs richtig handeln zu können.

4.1.1.4. Unternehmer

Die 47-jährige Bianca Keller ist eine sehr verantwortungsbewusste Geschäftsführerin eines bekannten Mittelständigen Unternehmens. Durch den Bekanntheitsgrad ihrer Firma, werden dessen Kunden vermehrt zum Ziel von Phishing Attacken. Die Reputation ihrer Firma und ihre Kunden, liegen Bianca sehr am Herzen. Es ist ihr deshalb wichtig, dass Sie sich für die Sicherheit ihrer Kunden einsetzt und diese über die Gefahren informiert oder davor bewahrt. Gleichzeitig möchte sie verhindern, dass ihr Unternehmen durch vermehrte Phishing Attacken einen Reputationsschaden erleiden könnte, da die Marke ihres Unternehmens missbraucht werden könnte.

4.1.2. Funktionale Anforderungen

Anforderungen können in funktionale und nichtfunktionale Anforderungen unterteilt werden. Kurz gesagt beschreiben funktionale Anforderungen, was ein System kann, nichtfunktionale, wie das System arbeitet. Im Folgenden werden funktionale Anforderungen über Use Cases beschrieben. Die funktionalen Anforderungen werden im casual Use Case Format [11] notiert und mit Unified Modeling Language (UML) visualisiert.

Abbildung 2 gibt einen Überblick über die Use Cases und die involvierten Akteure. Die Use Cases werden darauffolgend im Detail beschrieben.

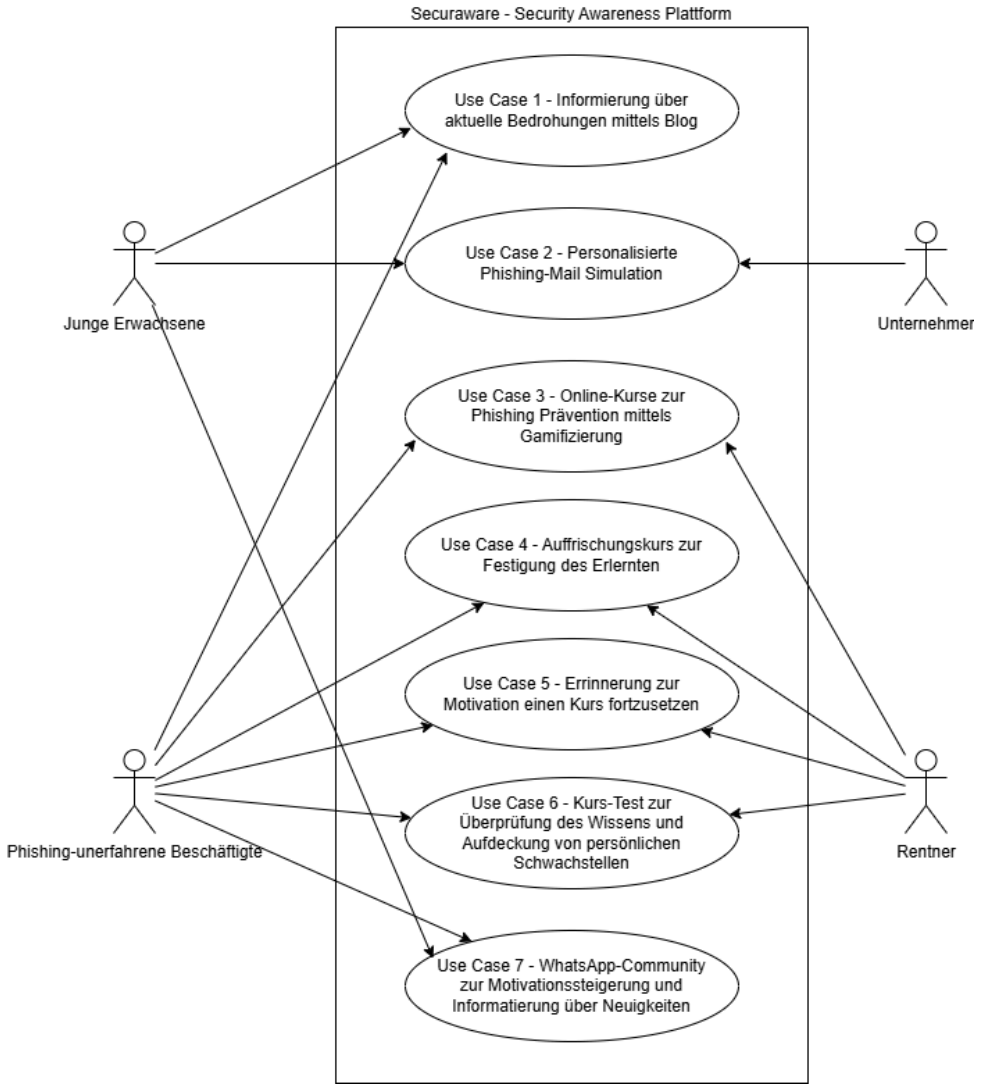


Abbildung 2: Use Case Diagramm

Da die Persona „Unternehmer“ in unserem Falle lediglich als Informationslieferant agiert und keinen Endnutzer der Plattform darstellt, liegt sein Interesse hauptsächlich im Use Case 2 „Personalisierte Phishing-Mail Simulation“.

Die restlichen Personas („Junge Erwachsene“, „Phishing-unerfahrene Beschäftigte“ und „Rentner“) können zwar an allen Use Cases interessiert sein und haben auch auf alle Use Cases Zugriff, wir zeigen in diesem Diagramm allerdings die, auf die entsprechenden Personas, am besten zutreffenden Use Cases auf.

4.1.2.1. Use Case 1 - Informierung über aktuelle Bedrohungen mittels Blog

Primärszenario Ein Benutzer der Plattform möchte sich über die aktuelle Bedrohungslage und aktuell häufig verwendete Angriffsmethoden informieren. Über den Blog auf der Webseite von Securaware werden ihm diese Informationen zur Verfügung gestellt.

4.1.2.2. Use Case 2 - Personalisierte Phishing-Mail Simulation

Primärszenario Ein Benutzer der Plattform meldet sich über die Webseite von Securaware für die personalisierte Phishing-Mail Simulation an. Das System wird ihm im Anschluss in unregelmässigen Abständen, auf seinen Wissensstand und sein bisheriges

Verhalten bei dieser Simulations-Funktion zugeschnittene „Phishing-Mails“ zu senden, welche jeweils mindestens einen Link enthalten. Erkennt der Benutzer die E-Mail nicht korrekt als Phishing, so wird dieser auf den Link klicken, über welchen er auf eine exakt für diese E-Mail zugeschnittene Webseite gelangt, auf welcher ihm aufgezeigt wird, wie er die E-Mail korrekt als Phishing-Mail hätte erkennen können. Das System wird ihm also das nächste Mal ein etwas leichter erkennbares Phishing E-Mail zusenden.

Sekundärszenario Sollte der Benutzer die E-Mail korrekt als Phishing-Mail einstufen, so wird dieser auch nicht auf den Link klicken. Das System geht dann nach ca. 1 Woche nach dem Öffnen der E-Mail davon aus, dass der Benutzer die E-Mail korrekt erkennt hat. Das System wird ihm das nächste Mal also ein etwas schwieriger zu erkennendes Phishing E-Mail zusenden.

4.1.2.3. Use Case 3 - Online-Kurse zur Phishing Prävention mittels Gamifizierung

Primärszenario Ein Benutzer der Plattform möchte erfahren, wie er sich am besten vor Phishing Attacken schützen kann. Dazu registriert er sich über die Webseite von Securaware, worin ihm anschliessend mögliche Online-Kurse angeboten werden, die der Benutzer durchspielen kann. Die Online-Kurse werden mit einer Gamifizierung umgesetzt, wodurch der Benutzer für seine durchgeführten Kurse belohnt wird und ein Anreiz geschaffen wird, weitere Kurse durchzuführen.

Sekundärszenario Sollte sich der Benutzer über eine längere Zeit keine Fortschritte mehr erzielt haben, kann über Aktionen in der Gamifizierung wieder ein Anreiz geschaffen werden, was den Benutzer dazu animiert, weitere Online-Kurse zu absolvieren.

4.1.2.4. Use Case 4 - Auffrischungskurs zur Festigung des Erlernten

Primärszenario Ein Benutzer der Plattform, der sich bereits selbst oder über die Online-Kurse von Securaware Wissen über die Prävention von Phishing-Attacken angeeignet hat, möchte sein Wissen auffrischen. Dazu meldet er sich mit seinem Benutzerkonto bei Securaware an und startet den angegebotenen Refresher. In diesem Refresher wird dem Benutzer eine grafische Repräsentation einer E-Mail resp. eines E-Mail Clients mit interaktiven Elementen geboten, worüber er sich über die einzelnen Bestandteile der E-Mail informieren kann und wie ihm diese helfen, Phishing E-Mails korrekt zu erkennen.

Sekundärszenario 6 Monate nach dem Absolvieren eines Kurses von Securaware werden die Benutzer per Mail darüber informiert, dass empfohlen wird, nun den Refresher zu nutzen um sein angeeignetes Wissen zu vertiefen.

4.1.2.5. Use Case 5 - Erinnerung zur Motivation einen Kurs fortzusetzen

Primärszenario Ein Benutzer der Plattform, der bereits einen Kurs zur Phishing Prävention begonnen hat, hat diesen nicht abgeschlossen und für mehr als eine Woche auch nicht fortgesetzt. Das System sendet ihm aus diesem Grund eine Erinnerung per E-Mail um ihn darauf aufmerksam zu machen.

Sekundärszenario Der Benutzer hat ein Konto erstellt und die Plattform danach nicht mehr verwendet. Damit dieser schnellstmöglich Securaware wieder besucht, wird ihm am darauffolgenden Tag eine Erinnerung gesendet.

4.1.2.6. Use Case 6 - Kurs-Test zur Überprüfung des Wissens und Aufdeckung von persönlichen Schwachstellen

Primärszenario Ein Benutzer der Plattform hat einen Kurs abgeschlossen und möchte nun deshalb sein erlerntes Wissen prüfen. Dafür wird ihm nun ein Kurs-Test angeboten, der das vermittelte Wissen in Form von Multiple-Choice Fragen und interaktiven Elementen abfragt. Der erfolgreiche Abschluss eines solchen Kurs-Tests wird mit einer höheren Belohnung in Form von Erfahrungspunkten belohnt.

Sekundärszenario Ein neuer Test wurde erstellt und freigeschalten. Die Benutzerin möchte ihr Wissen erneut testen und den Test abschliessen.

4.1.2.7. Use Case 7 - WhatsApp-Community zur Motivationssteigerung und Informierung über Neuigkeiten

Primärszenario Ein Benutzer der Plattform möchte nicht ständig die Seite nach neuen Lern-Inhalten oder Blog-Beiträge zu aktuellen Gefahren durchstöbern müssen. Stattdessen möchte er stets ohne grossen Aufwand auf dem Laufenden bleiben. Ihm wird dazu die Anmeldung zur Securaware WhatsApp-Community vorgeschlagen, wodurch er mittels Push-Nachrichten über neue Inhalte oder Beiträge informiert wird. Dem Benutzer werden in den Nachrichten direkte Links zu den neuen Inhalten geboten, damit diese einfach zugreifbar sind.

4.1.3. Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen beschreiben, wie ein System funktioniert. Sie legen Qualitätsmerkmale, Einschränkungen oder Rahmenbedingungen fest, die unter anderem das Verhalten, die Leistung oder die Benutzerfreundlichkeit betreffen.

Titel	**NFR1 Performance**
Beschreibung	Zeit, um ein Online-Kurs-Modul zu laden
Messbarkeit	Angenommen der Benutzer hat momentan die Webseite mit Modul A offen. Es darf maximal eine Sekunde dauern, bis die Website mit Modul B geladen ist.
Begründung	Empirische Evidenz zeigt, dass Lernkurse häufig nicht mit hoher Motivation abgearbeitet werden. Daher ist es entscheidend, die bestehende Motivation der Benutzer aufrechtzuerhalten. Ein nahtloser und schneller Webseitenwechsel von Modul A zu Modul B trägt massgeblich dazu bei.

Tabelle 1: NFR1 Performance

Titel	**NFR2 Security**
Beschreibung	Sicherheit und Datenschutz generell
Messbarkeit	• Benutzer haben ausschliesslich Zugriff auf ihre Funktionalität. Schnittstellen führen eine Authorisierung durch. • Passwörter werden nicht in Klartext gespeichert sondern nur deren Hashwert. Die Hashfunktion ist für das hashen von Passwörtern ausgerichtet. • Der Transfer von Daten zwischen dem Benutzer und Securaware ist verschlüsselt.
Begründung	Der Hauptzweck von Securaware ist, Benutzer auf das sicherheitsrelevante Problem Phishing zu sensibilisieren. Entsprechend soll die Software ihre eigene Sicherheit und Resilienz gegenüber Angreifern unter Beweis stellen. Zusätzlich sollen Benutzerdaten sensibel behandelt und persistiert werden.

Tabelle 2: NFR2 Security

Titel	**NFR3 Usability**
Beschreibung	Benutzererfahrung User Experience (UX)
Messbarkeit	Es wird mindestens eine Session von Usability-Tests durchgeführt. Es werden Erkentnisse aus den Resultaten der Tests gezogen und diese werden umgesetzt.
Begründung	Securaware ist ein Prototyp für ein Produkt. Entsprechend ist es von hoher Bedeutung, dass der Prototyp benutzerfreundlich ist, damit das daraus resultierende Produkt verwendet wird.

Tabelle 3: NFR3 Usability

Titel	**NFR4 Portability**
Beschreibung	Plattformunabhängiges Deployment
Messbarkeit	Die C4-Container [12] sind virtualisiert mit zum Beispiel Docker [13].
Begründung	Um eine kontinuierliche Testung der Software zu ermöglichen und das Feedback aus Usability-Tests effizient umzusetzen, soll das Deployment automatisiert werden. Zusätzlich soll es den Industriestandard folgen.

Tabelle 4: NFR4 Portability

Titel	**NFR5 Accessibility**
Beschreibung	Zugänglichkeit, Barrierefreiheit
Messbarkeit	Die Webpage erfüllt den Standard Web Content Accessibility Guidelines (WCAG) 2.1 [14] gemäss dem Level A.
Begründung	Securaware soll für möglichst viele Benutzerinnen und Benutzer zugänglich sein.

Tabelle 5: NFR5 Accessibility

4.2. Produkt

Dieses Unterkapitel beschreibt, was für Features wie umgesetzt werden. Der Fokus liegt hierbei auf dem effektiven Feature und dessen Wert. Das Technische wird in diesem Kapitel nicht miteinbezogen.

4.2.1. Online-Kurs

Der Online-Kurs wird in Module unterteilt. Dadurch können Informationen in absehbaren Lerneinheiten konsumiert werden. Zusätzlich kann somit der Fortschritt des einfacher nachverfolgt werden, wodurch spezifischere Erinnerungen (siehe Abschnitt 4.2.3) erstellt werden können. Im folgenden werden die Module anhand der Abbildung 3 zusammengefasst und anschliessend genauer beschrieben. Die Module sind auf deutsch verfasst. Die dunkelblauen Boxen repräsentieren die Modul-Thematiken und Titel, die Hellblauen die Inhalte.

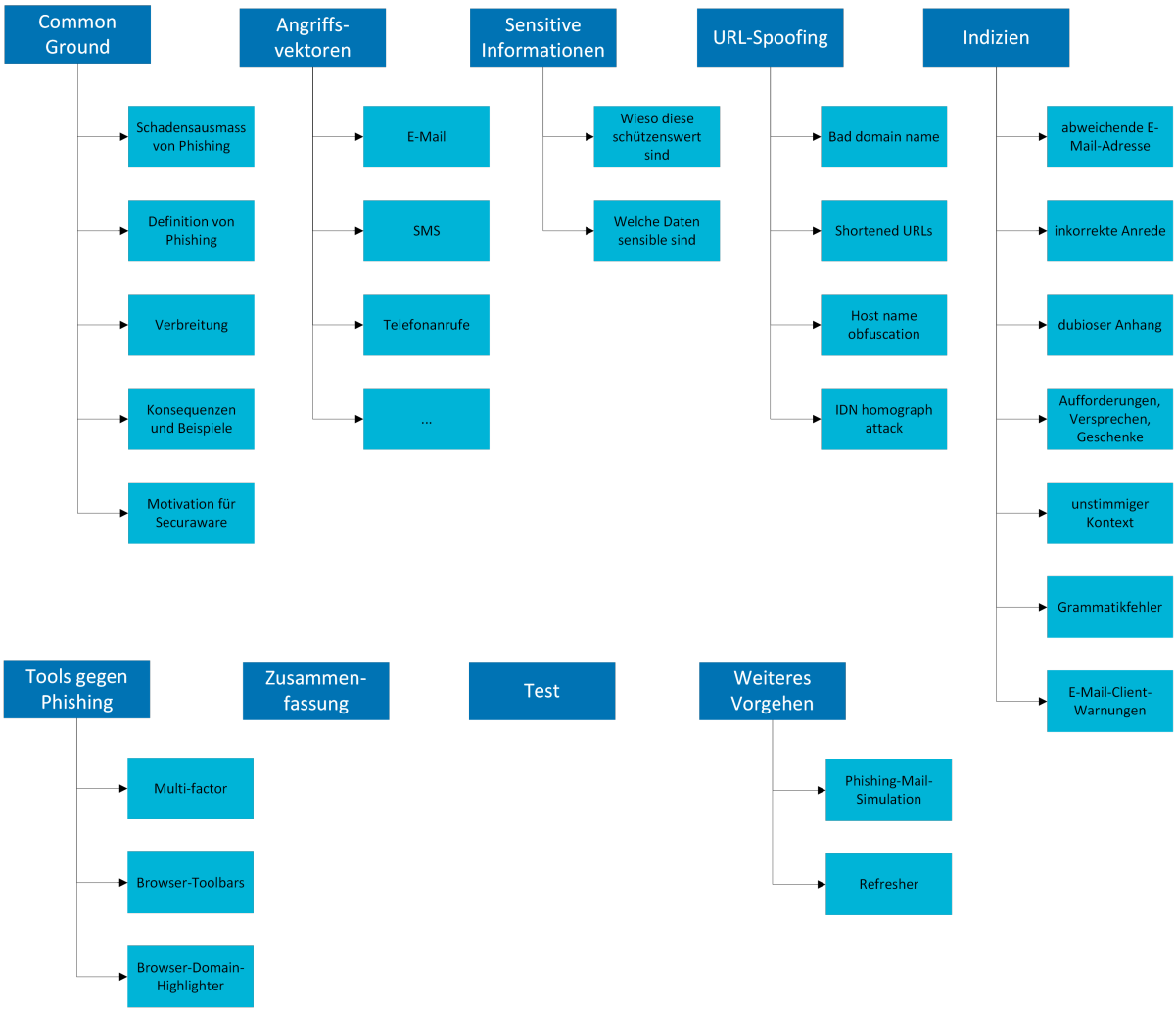


Abbildung 3: Modulthematiken und -Inhalte dee Online-Kurse

- **Common ground, Aufbau einer Grundlange**: Dem Benutzer wird in diesem Modul erklärt,
  - weshalb man sich vor Phishing schützen soll,
  - was Phishing ist,

    - dass **jeder** davon betroffen ist,
    - was für Konsequezen anfallen bei einem erfoglreichen Angriff anhand von einem prägnantem Beispiel und
    - wieso er oder sie die Kurse von Securaware belegen soll mitsamt einer kurzen Übersicht

In diesem Modul wird unter anderem dem Benutzer eine Motivation gegeben, sich über Phishing zu informieren und dies über Securaware zu tun.

- **Angriffsvektoren**: Es wird erklärt, über Welche Wege, Medien ein Angriff ausgeführt werden kann. Diese werden mit Beispielen erläutert. Es werden die Medien des Abschnitt 2.1.1 verwendet.
- **Sensitive Informationen**: Sensitive Informationen sind die kritischen Daten, die vor einem Phishing-Angriff geschützt werden sollen. Es wird erklärt,
    - wieso diese geschützt werden müssen beziehungsweise wie der Angreifer diese missbraucht und
    - was für Informationen sensitiv und daher schützenswert sind.
- **URL-Spoofing**: Dieses Modul verdäutlicht, wie eine URL verfälscht werden kann. Es werden die Techniken des Abschnitt 2.1.3.1 beschrieben. Es wird zusätzlich darauf hingewiesen, dass die Domain einer URL der ausschlaggebende Punkt ist.
- **Indizien**: In diesem Modul wird erläutert, wie die Benutzerin eine Phishing-Mail erkennen kann. Indizien für eine betrügerische E-Mail sind:
    - eine vom Absender abweichende E-Mail-Adresse
    - inkorrekte Anrede
    - Spoofed Links
    - Archivdateien (z.B. .zip) und ausführbare Programme (z.B. .exe) im Emailanhang
    - Aufforderungen
    - Versprechen
    - dubiose Gutschriften, Gutscheine oder anderweitige Geschenke
    - unstimmiger Kontext (siehe Abschnitt 2.1.2.2)
    - je nach Kontext Grammatikfehler
    - Warnungen, welche der E-Mail-Client anzeigt
    - Anleitungen
- **Tools gegen Phishing**: Es wird geschildert, welche technischen Tools dem Benutzer oder der Benutzerin dabei helfen, betrügerische E-Mails zu identifizieren. Es werden die Tools des Abschnitt 2.1.4.2 geschildert.
- **Zusammenfassung**: Dieses Modul fasst den Kurs zusammen und wiederholt die kritischen Informationen und Aspekte.
- **Kurs-Test**: Aufgrund der Grösse dieses Moduls wird es als eigenes Feature betrachtet und genauer in Abschnitt 4.2.4 beschrieben.

- **Weiteres Vorgehen**: In diesem Modul wird die Phishing-Mail-Simulation umrissen. Es wird beschrieben, wie diese Simulation den Benutzer weiter sensibilisiert auf Phishing-Angriffe.

4.2.2. Refresher

Ziel dieses Features ist, das Wissen des Benutzers hinsichtlich Phishing aufzufrischen. Dies wird durch einen dedizierten Kurs bewerkstelligt. Dieser ist erst 6 Monate, nachdem die Benutzerin das letze Modul abgeschlossen hat, verfügbar. Der Benutzer wird über eine E-Mail über den Refresher informiert. Der Refresher besteht aus einer Webseite, welche wie Microsoft Outlook aussieht, in welcher eine Phishing-E-Mail angezeigt wird. Die Benutzerinn kann interaktiv Elemente wie zum Beispiel den Absänder oder einen Link auswählen und so Information erhalten. Der Refresher soll kurz, prägnant und spielerisch sein.

4.2.3. Erinnerung

Der online Kurs muss nicht am Stück absolviert werden und kann zu einem späteren Zeitpunkt weiter verfolgt werden. Erinnerungen sollen dem Benutzer dabei helfen, den Kurs später fortzusetzen: Nach einer gewissen Zeit wird der Benutzer informiert und motiviert, den Kurs vollständig abzulegen.

4.2.4. Kurs-Test

Der Benutzer kann mit diesem Feature überprüfen, wie gut er oder sie den Online-Kurs verstanden hat und wo er oder sie noch Verbesserungspotential aufweist. Der Test umfasst alle Module und gibt eine dedizierte Bewertung pro Modul ab, damit Verbesserungspotential bestens lokalisiert werden kann.

4.2.5. Phishing-Mail-Simulation

Aktives, wiederkommendes Training sorgt dafür, dass das erlangte Wissen durch die Module nicht verloren geht und zusätzlich herausgefordert wird. Dieses Feature stellt solch ein Training zur Verfügung. In zufälligen Zeitabschnitten wird dem Benutzer ein Phishing-E-Mail gesendet. Dieses beinhaltet Täuschungsindizien und soll vom Benutzer entsprechend als Phishing-Angriff erkannt werden. Neben den Inidizien sind betrügerische Links in das E-Mail eingebettet. Wenn der Benuter diese auswählt, wird er auf eine Webseite weitergeleitet, welche die E-Mail als Phishinig-Angriff enttarnt. Nebenläufig wird Securaware darüber informiert, dass der Benutzer den Link ausgewählt hat. Dadurch kann Securaware den Schwierigkeitsgrad beeinflussen und den nächsten Phishing-Angriff einfacher zum enttarnen gestalten.

4.2.6. WhatsApp Community

Die Benutzerinnen und Benutzer sollen dazu motiviert werden, sich kontinuierlich mit dem Thema Phishing auseinanderzusetzen, anstatt sich einmalig und in kompakter Form, wie beispielsweise im Online-Kurs, damit zu beschäftigen. Eine nachhaltige Motivation kann insbesondere durch ein spannendes und alltagsnahes Format erreicht werden. Idealerweise erfolgt die Sensibilisierung über eine Plattform, die weit verbreitet ist und regelmässig genutzt wird. WhatsApp Communities erfüllen diese Anforderungen.

Über Benachrichtigungen in der Community können Nutzerinnen und Nutzer auf aktuelle Phishing-Angriffe aufmerksam gemacht werden oder es können Produkte von Securaware beworben werden. Im Vergleich zu anderen Kanälen wie E-Mail oder Blogs bietet WhatsApp eine persönlichere Ansprache. Dies liegt an der individuellen Nutzung von WhatsApp, da über die Plattform meist direkte, persönliche Nachrichten mit Kontakten ausgetauscht werden.

Für die Umsetzung dieses Konzepts muss zunächst die technische Machbarkeit überprüft werden. Damit Nachrichten automatisiert in eine benutzerdefinierte WhatsApp-Community gesendet werden können, ist eine entsprechende Application Programming Interface (API) von WhatsApp beziehungsweise dessen Entwickler Meta erforderlich. Diese API existiert, ist aber nur für Unternehmungen zugänglich. Entsprechend wird dieser Ansatz verworfen.

Es gibt Drittanbieter, die vergleichbare Schnittstellen anbieten. Allerdings erfordern diese den vollständigen Zugriff auf den WhatsApp-Account, einschliesslich aller Kontakte, Bilder und Nachrichten, auf die auch WhatsApp selbst Zugriff hat. Aufgrund der damit verbundenen Datenschutzrisiken ist dieser Ansatz nicht tragbar. Daher wird die Nutzung einer Drittanbieter-API verworfen.

Entsprechend kann das Senden von Nachrichten in eine WhatsApp Community nicht automatisiert werden. Die Nachrichten können manuell gesendet werden.

4.2.7. Phishing Mentor

Benutzer, welche nicht affin gegnüber Phishing sind, sind sich der Gefahr von Phishing nicht bewusst. Entsprechend ist die Motivation sich mit Phishing auseinanderzusetzen eher gering. Dabei sind es genau diese Benutzer, welche am anfälligsten auf Phishing-Angriffe sind und am meisten von Securaware profitieren.

Der Phishing Mentor sieht vor, dass ein Benutzer A Phishing-E-Mails oder Kürse für einen nicht-Phishing-affinen Benutzer B auswählt. Nach ein wenig vergangener Zeit erhält Benutzer B die entsprechende Simulations-Email oder eine Kurs-Empfehlung. Sommit nimmt Benutzer A eine Lehrerposition wahr und motiviert Benutzer B sich mit Phishing auseinanderzusetzen.

Dieses Feature kann zum Beispiel für Familien nützlich sein: Angenommen der Grossvater ist nicht affin gegenüber Phishing, seine Tochter könnte ihn über Securaware schulen.

Dieses Feature existiert gemäss Recheren nicht in vergleichbaren Produkten und ist somit ein „unique selling point“, mit dem sich Securaware von anderen Plattformen abhebt.

4.3. Technisch

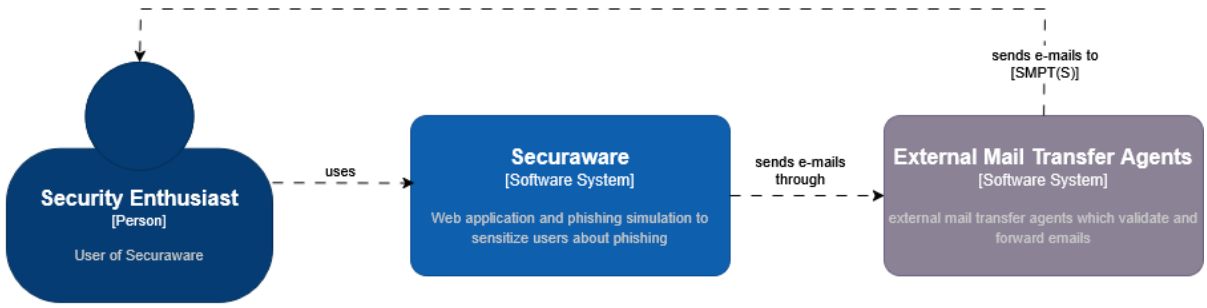
In diesem Unterkapitel wird beschrieben, wie das Produkt technisch umgesetzt wird. Die strategische, abstrakte Lösung ist der Fokus dieses Unterkapitels. Technische Details werden in kommenden Kapiteln behandelt.

4.3.1. Software-Architektur

Die Architektur wird gemäss dem C4-Modell [12] dokumentiert.

4.3.1.1. Context

Das Systemkontext-Diagramm zeigt die Interaktion zwischen dem System und den externen Akteuren. Alle Benutzer und Stakeholder von Securaware sind im Folgenden als eine Person zusammengefasst. Die Benutzer interagieren mit dem System, um sich in Phishing auszubilden. Securaware interagiert mit externen mail transfer agents, welche E-Mails validieren und weiterleiten. Dieser Context kann vereinzelte E-Mails verzögern oder in Extremfällen nicht ausstellen.

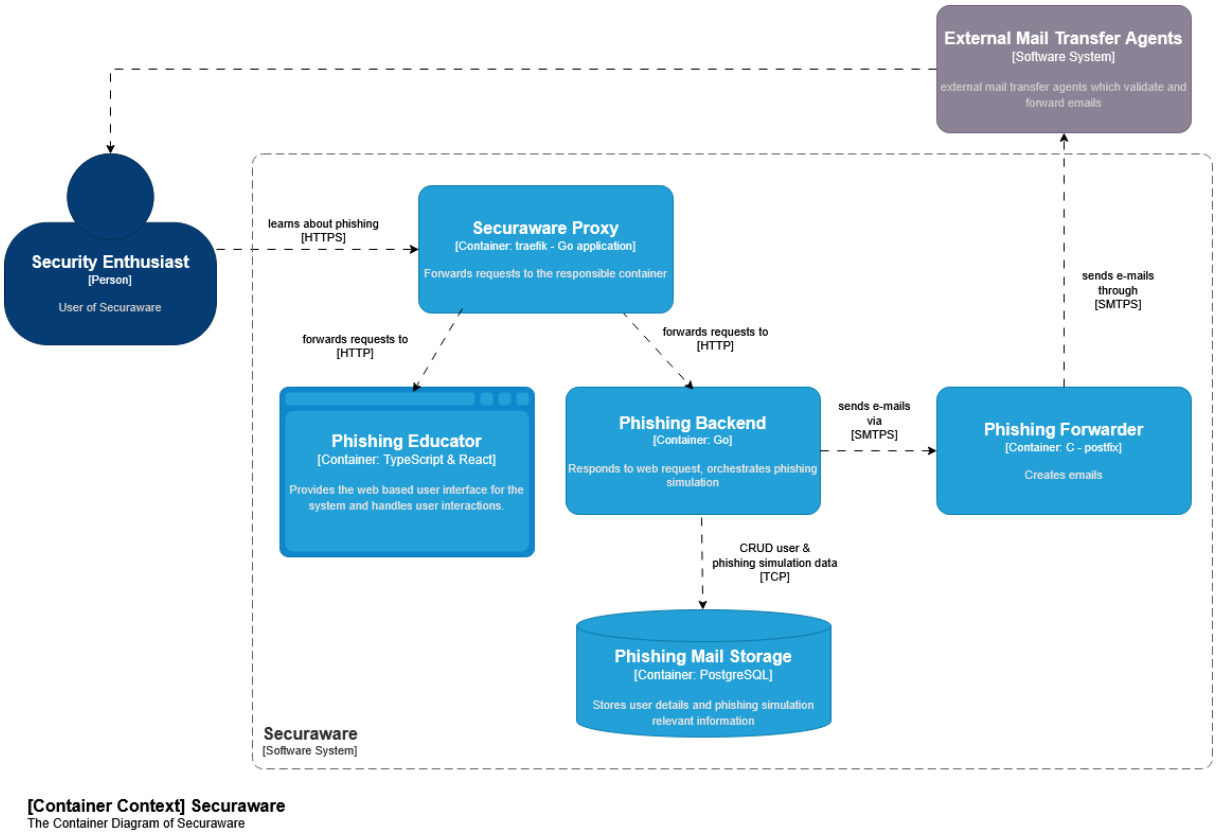


[System Context] Securaware
The Context Diagram of Securaware

Abbildung 4: C4 Context-Diagramm von Securaware

4.3.1.2. Container

Securaware wird als Webapplikation realisiert, sodass der Benutzer keine Software installieren muss und sämtliche Betriebssysteme unterstützt werden. Webanfragen werden vom Securaware Proxy angenommen und an den entsprechenden Container weitergeleitet. Der Securaware Proxy dient als Vermittler, sodass der Benutzer nur mit diesem und dessen IP-Adresse kommuniziert. Dieser leitet Webanfragen an den Phishing Educator und das Phishing Backend weiter. Der Phishing Educator stellt die graphische Benutzeroberfläche zur Verfügung. Das Phishing Backend hingegen orchestriert die Phishing Simulation, beantwortet Webanfragen bezüglich benutzerspefischen Aspekten und interagiert mit dem Phishing Mail Storage. Der Phishing forwarder stellt E-Mails aus. Die Phishing Mail Storage persistiert Daten.



[Container Context] Securaware
The Container Diagram of Securaware

Abbildung 5: C4 Container-Diagramm von Securaware

Die Container werden genauer mit Y-Templates [15] beschrieben.

Phishing Educator

In the context of	NFR1 Performance (siehe Tabelle 1)
facing	the need to load web based learning modules fast
we decided	to build the web app as a single page application
and neglected	a Static Site Generation (SSG) approach
to achieve	an even faster switch from module to module and to lead dynamicaly content
accepting the downside	that the first web page that the user loads will take more time

Tabelle 6: Phishing Educator y-Template about performance

Securaware Proxy

In the context of	the web based architecture of Securaware
facing	the limitation of having only one single IPv4 address available in production
we decided	to introduce a proxy
and neglected	to run both web backend and frontend based concerns in the same container
to achieve	seperation of concerns, ease of development and flexability
accepting the downside	that a new container, the proxy, is introduced

Tabelle 7: Securaware Proxy y-Template about IPv4 addresses

In the context of	the web based architecture of Securaware
facing	the limitation of having only one single IPv4 address available in production
and neglected	to use ESNI as both Google Chrome and Mozilla Firefox only support it by setting a specific flag [16]

Tabelle 8: Securaware Proxy y-Template about ESNI

Phishing Backend

In the context of	ease of development
facing	the need to be productive to finish Securaware on time
we decided	to use Golang as programming language
and neglected	Java and the powerful webframework Spring
to achieve	a rapid and easier development compared to using Spring
accepting the downside	Golang is less mature than Java

Tabelle 9: Phishing Backend y-Template about programming language

In the context of	database access and management
facing	the urge of ease of development
we decided	to use an ORM
and neglected	plain SQL commands
to achieve	a rapid and easier development
accepting the downside	of the dependency to an external library and its limitations and learning curve

Tabelle 10: Phishing Backend y-Template about the usage of an ORM

In the context of	the folder structure of the code
facing	the need to have a comprehensible folder structure which in addition follows an industry standard
we decided	to put the main method into the „cmd“ folder, automated integration tests into the „integration_tests“ folder and all the rest into the go specific „internal“
and neglected	to apply the functional decomposition to the root of the project
to achieve	a go compliant folder structure
accepting the downside	that the functional decomposition pattern is not fully implemented

Tabelle 11: Phishing Backend y-Template about folder structure

In the context of	HTTP server development
facing	the need to use mature APIs
we decided	to use Golang's standard library
and neglected	external libraries
to achieve	a rapid development without having to learn a web framework as the standard library is very mature
accepting the downside	that some HTTP related code must be written manually

Tabelle 12: Phishing Backend y-Template about using a web framework

In the context of	descriptive API design
facing	the need to have a well described API
we decided	to use OpenAPI as an HTTP interface definition language
to achieve	a well described API so that on both the frontend and backend can be worked on in parallel
accepting the downside	the OpenAPI document must be maintained

Tabelle 13: Phishing Backend y-Template about how to describe the API

In the context of	ease of development
facing	the opportunity to simplify development and to ensure API compliance
we decided	to generate code for the HTTP request and response body out of the API
and neglected	to write the code manually
to achieve	a simplified development and to ensure API compliance
accepting the downside	of the dependency to an external library

Tabelle 14: Phishing Backend y-Template about generating code out of the API

In the context of	error handling
facing	the need to have comprehensible errors
we decided	to implement the RFC 9457 [17]
and neglected	to only use HTTP status codes as a mean of information
to achieve	a comprehensible errors
accepting the downside	that some code generation functionality can not be used

Tabelle 15: Phishing Backend y-Template about error handling

In the context of	configurations
facing	the need to have flexible configurations for testing and local development
we decided	to use environment variables
and neglected	to use .yaml, .toml, .xml and other ways to store configurations
to achieve	a programming and operating system independent way of applying configurations
accepting the downside	that no type safety is given

Tabelle 16: Phishing Backend y-Template about error handling

In the context of	monitoring the runtime behaviour
facing	the need to backtrack errors and bugs
we decided	to use structured JSON logging
and neglected	to use unstructured logging
to achieve	an easy way to find logs
accepting the downside	that some configuration is needed to set up the structured logging

Tabelle 17: Phishing Backend y-Template about monitoring the runtime behaviour

Phishing Forwarder

In the context of	risk risk management
facing	the need to send emails with an non university IP address
we decided	to create a dedicated component that sends out the phishing simulation related emails
and neglected	a third party mail service
to achieve	more control over the emails and their meta data
accepting the downside	a new dedicated container has to be created and maintained

Tabelle 18: Phishing Forwarder y-Template about risk management

Phishing Mail Storage

In the context of	storing data
facing	the need to store data in a manitainable way with speed improvements options such as indexes
we decided	to use a relational database
and neglected	non-relational databases
to achieve	a stable data model and easy to improve performance
accepting the downside	that setting up and changing the schema is more difficult

Tabelle 19: Phishing Mail Storage y-Template about SQL vs NoSQL

In the context of	ease of development
facing	the need to be productive to finish Securaware on time
we decided	to use PostgreSQL as relational database
and neglected	other relational databases such as MySQL, Oracle Database or MariaDB
to achieve	a rapid development, a stable application with a great community support and no dependency to commercial products
accepting the downside	that other relational databases are slightly faster

Tabelle 20: Phishing Mail Storage y-Template about PostgreSQL

In the context of	the format how data is stored
facing	the need to find the optimum between read and write efficiency
we decided	to use mainly the 3rd normal form
and neglected	the other normal forms
to achieve	the optimum between read and write efficiency
accepting the downside	data could either be stored more read or write friendly

Tabelle 21: Phishing Mail Storage y-Template about normalization forms

4.3.1.3. Component

Die Component-Schicht listet Components und die Interaktion zwischen diesen auf. Aufgrun der Grösse der Container wird auf die Visualisierung dieser Schicht verzichtet.

4.3.1.4. Code

Die Code-Schicht zeigt detailliert Softwareprimitiven wie zum Beispiel, Klassen, Interfaces, Structs oder Prototypen und ihre Beziehungen innerhalb der entsprechenden Komponente auf. Diese Schicht ist sehr detailliert und wiederspielt den Code sehr genau. Entsprechend oft wird diese angepasst. Aufgrund dieser hohen Volatilität wird auf das Code-Repository verwiesen und auf die Visualisierung dieser Schicht verzichtet.

4.3.2. Tech-Stack

In diesem Unterkapitel werden die verwendeten Technologien aufgelistet und begründet, weshalb genau diese verwendet werden.

4.3.2.1. Phishing Educator

Da der Phishing Educator eine Single Page Web Application ist, wird selbstsverständlich HTML und Cascading Style Sheets (CSS) verwendet. Anstelle von JavaScript wird TypeScript verwendet, da dies Typen und compile time checks hinzufügt.

Gemäss Industriestandard und eigener Erfahrung wird ein Framework verwendet, das die Entwicklung vereinfacht und Kernfunktionalitäten wie zum Beispiel Templating anbietet. Es werden die nun vier modernen Industrievorreiter gemäss eigener Erahrung verglichen in Tabelle 22. jQuery [18] wird nicht miteinbezogen aufgrund der geringen Mächtigkeit und seines in die Jahre gekommenen Ansatzes. Astro [19] wird nicht miteinebezogen aufgrund der geringen Mächtigkeit.

Mit „Vertrautheitsgrad“ ist gemeint, wie viel persönliche Erfahrung mit dem entsprechenden Framework vorhanden ist und wie fortgeschritten der Kenntnisstand damit ist. Mit „Vorhandene Komponenten-bibliotheken“ ist gemeint, wie häufig eine Komponentenbibliothek für das entsprechende Framework vorhanden ist. Die Bewertung dieses Kriterium basiert auf persönlichen Erfahrungen. Die „Verbreitung“ wird gemäss den Sternen im entsprechenden Git-Repository gemessen.

	AngularJS [20]	**React** [21]	**Vue.js** [22]	**Svelte** [23]
Vertrautheitsgrad	3	4	1	1
Vorhandene Komponenten-bibliotheken	3	4	2	1
Verbreitung	3 (97k)	4 (234k)	2 (50k)	3 (82k)
Σ	**9**	**12**	**5**	**5**

Tabelle 22: Vergleich JavaScript-Frameworks und Libraries. Bewertung [0-4], wobei 0 die tiefste und 4 die höchste Bewertung ist.

React [21] weist die grösste Punktzahl auf und wird entsprechend als Framework verwendet.

4.3.2.2. Phishing Backend

Wie beim Phishing Educator, ist mit „Vertrautheitsgrad“ gemeint, wie viel persönliche Erfahrung mit dem entsprechenden Framework vorhanden ist und wie fortgeschritten der Kenntnisstand damit ist. Mit „Webserverorientierung“ ist gemeint, wie stark die Standardbibliothek die Erstellung eines Webservers vereinfacht. Damit geht die Entwicklererfahrung einher, also wie effizient Features erstellt werden können. Das Kriterum wird doppelt gewichtet. Die Bewertung dieses Kriterium basiert auf persönlichen Erfahrungen. Das Kriterium „Performance“ basiert den Benchmarks dieser Quellen [24], [25].

	JVM basierte Sprachen [26]	**.Net basierte Sprachen** [27]	**Go** [28]	**Rust** [29]	**Node.js** [30]
Vertrautheitsgrad	4	2	4	1	3
Webserver-orientierung	4	4	8	6	6
Performance	2	2	3	4	0
Σ	**10**	**8**	**15**	**11**	**9**

Tabelle 23: Vergleich Programmiersprachen für das Phishing Backend.

Go [28] weist die grösste Punktzahl auf und wird entsprechend als Programmiersprache verwendet. Zusätzlich wurde go spezifisch für cloud native Applikationen designed. cloud native landscape von CNCF, bestätigt dies, Go ist die meist verbreitete Sprache in diesem Landscapte [31].

4.3.2.3. Restlichen Container

Die restlichen Container verwenden existierende Software, welche in Abschnitt 5 beschrieben werden. Der Tech-Stack der verwendeten Software ist ein

Implementationsdetail der Software, entsprechend nicht relevant und wird nicht weiter erläutert.

5. Technische Umsetzung

In diesem Kapitel wird die technische Umsetzung von Securaware erläutert. Unter anderem wird die Umsetzung der in Abschnitt 4.3.1.2 definierten C4-Container in diesem Kapitel beschrieben.

5.1. HTTP-Schnittstelle

Der Phishing Educator und das Phishing Backend kommunizieren miteinander über HTTP. Die Schnittstelle an sich, deren Aufbau und markante Entscheidungen werden in diesem Kapitel beschrieben.

5.1.1. OpenAPI

Die Schnittstelle wird über die Interface Definition Language (IDL) OpenAPI [32] beschrieben. Diese ist eine in der Industrie weit verbreitete IDL . Durch die standardisierte Schnittstellenbeschreibung mittels OpenAPI wird die parallele Entwicklung von Frontend und Backend ermöglicht. Ein weiterer Vorteil ist die automatisierte Code-Generierung auf Basis der Schnittstellendefinition, welche detailliert in Abschnitt 5.3.5 beschrieben wird.

5.1.2. RESTful HTTP

RESTful HTTP stellt ein Paradigma dar, das Webdiensten spezifische Eigenschaften auferlegt, um den Anforderungen des World Wide Web gerecht zu werden [33]. Im Kontext von Securaware liegt der Fokus auf zwei zentralen Prinzipien dieses Paradigmas: der **einheitlichen Schnittstelle** und der **Zustandslosigkeit**.

Die einheitliche Schnittstelle manifestiert sich in der strukturierten Definition von Endpunkten und Ressourcen, die konsequent den REST-Prinzipien folgt. Die Zustandslosigkeit wird bewusst verfolgt, um die Komplexität der Anwendung zu reduzieren und eine hohe Skalierbarkeit zu gewährleisten.

Die Einhaltung der Prinzipien der einheitlichen Schnittstelle kann anhand des Richardson Maturity Model evaluiert werden [34]. Dieses Modell klassifiziert HTTP-Schnittstellen in vier Reifegrade (Level 0 bis 3), wobei ein höherer Level eine stärkere Adhärenz zu den REST-Prinzipien indiziert, siehe Abbildung 6. Ein hohes Level ist mit einer hohen Einhaltung gleichzusetzen.

Securaware wird das Level 2 dieses Modells unterstützen. Eine Implementierung von Level 3 wird aufgrund des signifikant grösseren Implementationsaufwandes nicht weiter verfolgt.

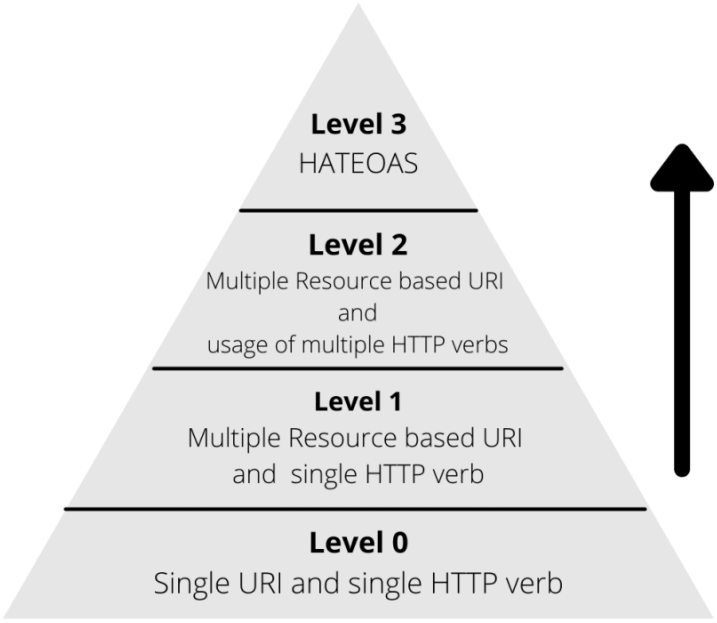


Abbildung 6: Beschreibung der REST Maturity Levels. Übernommen aus [1]

5.1.3. HTTP-Error-Handling

Die Fehlerbehandlung stellt einen essenziellen Bestandteil des Software-Engineerings dar. Eine konsistente Beschreibung und Rückgabe von Fehlern ist daher von grosser Bedeutung, um die Robustheit und Wartbarkeit von Softwaresystemen zu gewährleisten. Anstatt einen proprietären Standard zu entwickeln, wird in dieser Arbeit auf einen etablierten und fundierten Standard zurückgegriffen, nämlich dem RFC 9457 [17].

Angenommen der Benutzer möchte ein Profil erstellen. Hierbei verwendet dieser eine Email, welche bereits existiert. Die Schnittstelle gibt folgende Antwort zurück:

```
HTTP/1.1 422 Unprocessable Content
Content-Type: application/problem+json

{
  "title": "Diese Email wird bereits verwendet",
  "type": "urn:securaware:error:email-already-used"
  "status": 422
}
```

Code 1: Beispiel-RFC-9457-Fehlermeldung

Diese Fehlermeldung beinhaltet einen Titel, welcher dem Benutzer angezeigt werden kann, einen Bezeichner (type), um die Fehlermeldung von anderen zu unterscheiden und den HTTP-Status-Code. Letzterer wird im Body zusätzlich mitgegeben, da Router, Gateways und weitere Vermittler diesen im Body nicht modifizieren.

5.2. Phishing Educator

Nachfolgend wird die Umsetzung des Frontends von Securaware beschrieben, welches den Namen Phishing Educator trägt. Dabei handelt es sich um eine mit React umgesetzte

Benutzeroberfläche die den Zugriff auf Lerninhalte und Informationen zum individuellen Fortschritt ermöglicht.

5.2.1. Externe Softwarebibliotheken

In diesem Unterkapitel werden sämtliche im Phishing Educator eingesetzten externen Abhängigkeiten mit deren Verwendungszweck erläutert.

Bibliothek	**Verwendung**
react	Mächtige von Meta erstellte populäre Bibliothek, mit welcher effizient Benutzeroberflächen mit Komponenten erstellt werden können. Wird wegen noch mangelnder Unterstützung durch die verwendete Komponentenbibliothek in der Version 18 anstelle der neusten Version 19 verwendet.
react-dom	Einstiegspunkt für das Document Object Model (DOM) und React.
react-router	Von Shopify entwickeltes React Framework mit Routing Unterstützung
@react-router/node	Node-spezifische Werkzeuge und Hilfen für die React-Router-Integration auf dem Server und die Entwicklungsumgebung
@react-rouer/dev	Entwickler-Tools und Hilfsfunktionen für die Arbeit mit React Router
vite	Build-Tool zur Kompilierung des Typescript Codes
vite-plugin-cjs-interop	Vite-Plugin zur bessern Unterstützung von CommonJS-Modulen in ESM-Projekten
typescript	Bietet typsicheres JavaScript mit statischer Typisierung
sass	Präprozessor um den Umgang mit CSS zu erleichtern
@hey-api/openapi-ts	Codegenerator, welcher aus einer OpenAPI-Spezifikation einen Typescript-Typen sowie ein passendes Software Development Kit (SDK) erzeugt.
@hey-api/client-fetch	API-Client, der die Verwendung des zuvor erzeugten SDKs vereinfacht.
@fluentui/react-components	Komponentenbibliothek
@fluentui/react-icons	Icons der Komponentenbibliothek
apexcharts	Interaktive JavaScript-Diagrammbibliothek für visuelle Grafiken.
html-react-parser	Erlaubt das konvertieren von HTML -Strings in React-Elemente mit optionalem Filtern oder Umwandeln.
react-helmet-async	Asynchrone Implementation von React Helmet zur dynamischen Manipulation von <head>-Elementen.
react-apexcharts	Adapter zur Einbundung von apexcharts in React.

Bibliothek	Verwendung
framer-motion	Animationen für React
isbot	Identifiziert Bots, Crawlers und weitere automatisierte Prozesse anhand des User-Agent-Strings
cross-env	Plattformübergreifendes Setzen von Umgebungsvariablen in npm-Skripten
eslint	Tool zur statischen Codeanalyse und Einhaltung von Code-Stilrichtlinien.
@eslint/js	ESLint-Konfigurationspaket für moderne JavaScript-Standards
eslint-plugin-react-hooks	ESLint-Regeln zur Einhaltung der Regeln für React Hooks
globals	Sammlung globaler Variablennamen für verschiedene JavaScript-Umgebungen
typescript-eslint	Integration von TypeScript in ESLint zur statischen Analyse
tsx	All-in-One-Compiler für TypeScript und React

Tabelle 24: Externe Softwarebibliotheken des Phishing Educators

5.2.2. Client-side Routing

Für die Navigation innerhalb der Anwendung wird React Router als Framework verwendet. Die Entscheidung dafür wird aufgrund der grossen Popularität und der hohen Flexibilität getroffen, mit der das Routing gezielt an die Grösse und Komplexität der Applikation angepasst werden kann. So kann mit wenig Komplexität gestartet werden und falls nötig mit überschaubarem Aufwand auf ein komplexeren Routing-Modus gewechselt werden.

React Router wird in drei verschiedenen Modi angeboten:

- **Declarative Mode**: In diesem Modus wird die Routenstruktur vollständig deklarativ innerhalb der React-Komponenten definiert. Dieser Ansatz wird bevorzugt, wenn einfache Routing-Anforderungen bestehen oder wenn das Routing eng mit der UI verknüpft ist.
- **Data Mode**: Hier wird zusätzlich zur Routenstruktur auch die Datenbeschaffung in die Routenbeschreibung integriert (z.B. über loader und action). Dieser Modus wird eingesetzt, wenn ein zentraler Ort für Daten- und Navigationslogik benötigt wird - typischerweise bei mittleren bis grossen Applikationen.
- **Framework Mode**: In diesem Modus wird React Router als vollständig integrierter Teil einer umfassenden Applikationsstruktur verwendet. Dabei wird das Routing eng mit Mechanismen wie Datenbeschaffung, Formularverarbeitung und optional auch serverseitigem Rendering verbunden. Dieser Ansatz wird eingesetzt, wenn eine enge Verzahnung von Routing- und Anwendungslogik erforderlich ist - etwa bei umfangreichen prodktionsreifen Anwendungen mit hohen Anforderungen an Struktur, Wartbarkeit und Performance.

Durch diese klare Trennung der Modi kann React Router sowohl in kleinen, UI-zentrierten Komponenten, als auch in grossen, datengetriebenen oder serverseitig gerenderten Applikationen verwendet werden. Der jeweils passende Modus wird gewählt, ohne das zugrunde liegende Routing-Konzept oder die API-Struktur grundlegend ändern zu müssen.

Für diese Applikation wird der Framework Mode gewählt, da er die beste Trennung zwischen Routing, Datenbeschaffung und UI ermöglicht. Durch den modularen Aufbau können Routen, loader-Funktionen, actions und UI-Komponenten in einzelnen Funktionen definiert werden, welche durch Route Module miteinander verknüpft werden können. Dadurch wird eine klare Struktur geschaffen, die die Wartbarkeit und Lesbarkeit der Anwendung deutlich verbessert. Die Route-Definitionen können damit vollständig von der Datenbeschaffungs-Logik getrennt werden.

5.2.3. Hey API

Hey API kann anhand von OpenAPI-Beschreibungen TypeScript-HTTP-Client-Code generieren. Die in Abschnitt 5.1.1 beschriebene Schnittstelle wird hierfür verwendet. Code 2 zeigt auf, wie Hey-API für diesen Zweck konfiguriert werden kann.

```
1  import { createClient } from "@hey-api/openapi-ts";
2
3  try {
4    await createClient({
5      input: "../api/phishing-backend-open-api.yaml",
6      output: "./app/api",
7      plugins: ["@hey-api/client-fetch"]
8    })
9  } catch (e) {
10   console.error("Error generating the API client", e);
11 }
```

Code 2: HTTP-Client-Code-Generierung mit Hey API

- Zeile 5: Referenziert die OpenAPI-Spezifikation.
- Zeile 6: Referenziert den Output-Pfad, in welchem der generierte Code gespeichert werden soll.
- Zeile 7: Konfiguriert, dass die native JavaScript-fetch-API für HTTP-Anfragen verwendet werden soll.

5.3. Phishing Backend

Im folgenden wird die technische Realisierung des Backends von Securaware erläutert. Dieses wird in Go implementiert und übernimmt zentrale Aufgaben wie die serverseitige Logik sowie die Bereitstellung von Schnittstellen zur Kommunikation mit dem Phishing Educator.

5.3.1. Funktionale Dekomposition

Folgende konventionelle Architekturmuster werden für das Phishing Backend in Betracht gezogen:

Onion Architecture: Die Onion Architecture ist in Ringe unterteilt. Hierbei ist der innerste Ring, derjenige, der sich am wenigsten verändert. Abhängigkeiten verlaufen immer von aussen nach innen, äussere Ringe sind von Inneren abhängig aber Innere niemals von Äusseren. Anbindungen zu externen Systeme sind Bestandteil des äussersten Ringes. Innere Ringe definieren Interfaces, welche Äussere implementieren. Durch diese Interfaces können externe Syteme kostengünstig ausgetauscht werden. In einigen Anwendungsfällen besteht die Onion Architecture aus drei Ringen, nämlich dem Domain-Model-, Application- und Infrastruktur-Ring. Der Domain-Ring enthält das Domain-Model, der Application-Ring die Businessfunktionalität, um Use Cases zu implementieren und der Infrastruktur-Ring Code zum implementieren von Schnittstellen zu externen Systemen, wie zum Beispiel einer Datenbank- oder Hypertext Transfer Protocol (HTTP) -Schnittstelle. [2]

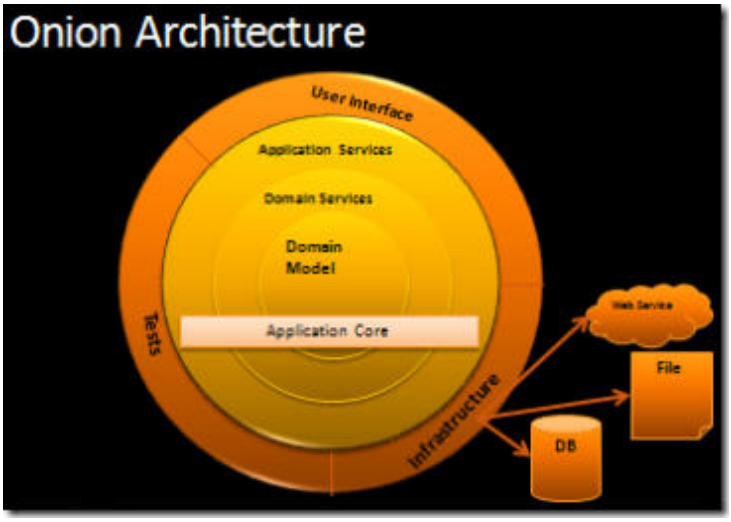


Abbildung 7: Beispiel einer Onion Architecture. Übernommen aus [2]

Clean Architecture: Die Clean Architecture verfolgt dieselben Prinzipien wie die Onion Architecture: Äussere Ringe sind von Inneren abhängig, innere Ringe sind niemals von Äusseren abhängig. In vielen Beispielen werden auch dieselben Ringe verwendet. Ein markanter Unterschied ist die Struktur des Application-Ringes: Während in der Onion Architecture dieser Ring in Services unterteilt wird, wird dieser in Clean Architecture in Features unterteilt. [35]

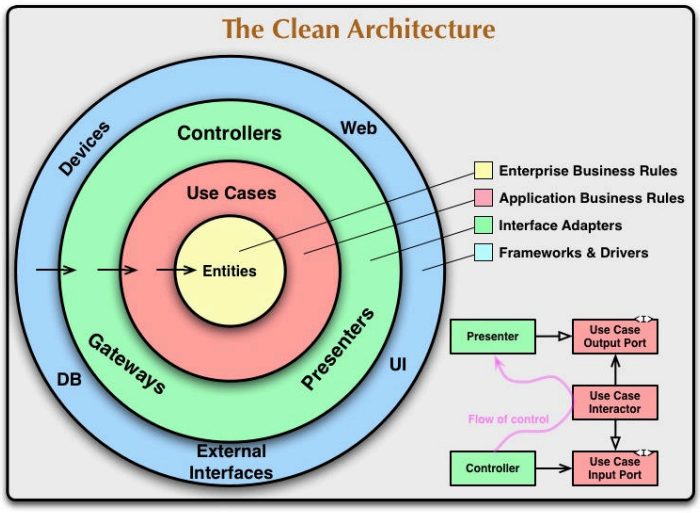


Abbildung 8: Beispiel einer Clean Architecture. Übernommen aus [3]

Hexagonal Architecture / Ports and Adapters: Die Hexagonal Architecture trennt den Anwendungskern, die Domäne, von externen Systemen. Die Kommunikation mit externen Systemen erfolgt über Schnittstellen , Ports. Adapter implementieren diese Ports. Durch diese Trennung können mehrere Adapter gleichzeitig betrieben, überarbeitet und ausgetauscht werden. Entsprechend eignet sich diese Architektur, wenn viele externe Systeme eingebunden werden. [36]

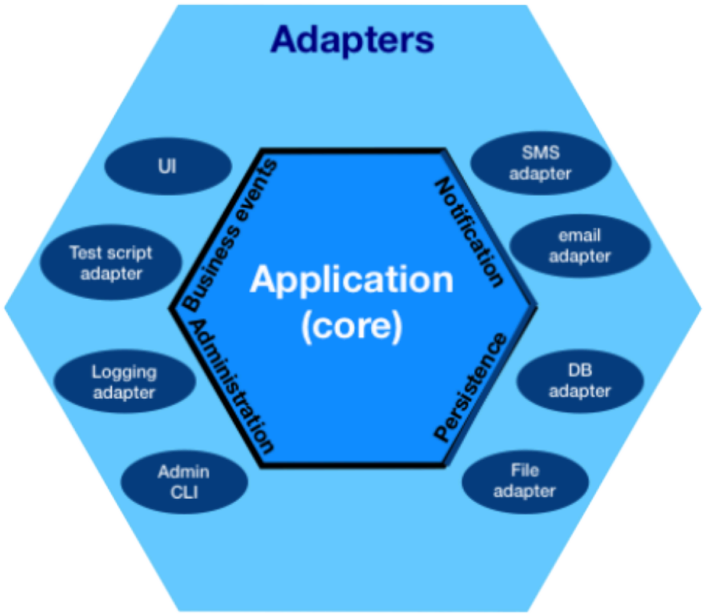


Abbildung 9: Beispiel einer Hexagonal Architecture. Übernommen aus [4]

Layered Architecture: Die Layered Architecture unterteilt eine Anwendung in technische Schichten. Diese Schichten sind hierarchisch gegliedert und dürfen jeweils nur mit der direkt darunterliegenden Schicht kommunizieren. In vielen Fällen sind viele bis hin zu alle Schichten von externen Systemen wie zum Beispiel der Infrastruktur abhängig. [37]

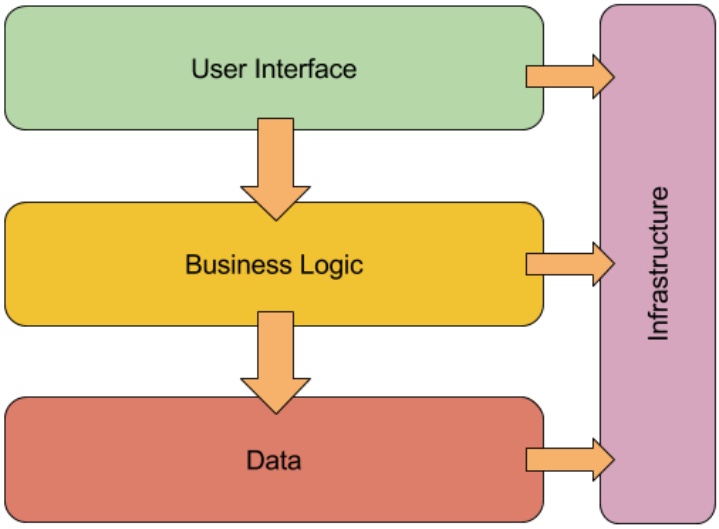


Abbildung 10: Beispiel einer Layered Architecture. Übernommen und angepasst nach [3]

Vertical Slice Architecture: Die Vertical Slice Architecture ist das Contraire zur Layered Architecture; Sie unterteilt eine Anwendung in funktionale Einheiten anstatt in technische Schichten. Dadurch wird eine hohe Kohäsion gewährleistet, Code für ein spezifisches Feature is somit über wenige Dateien verteilt. [5]

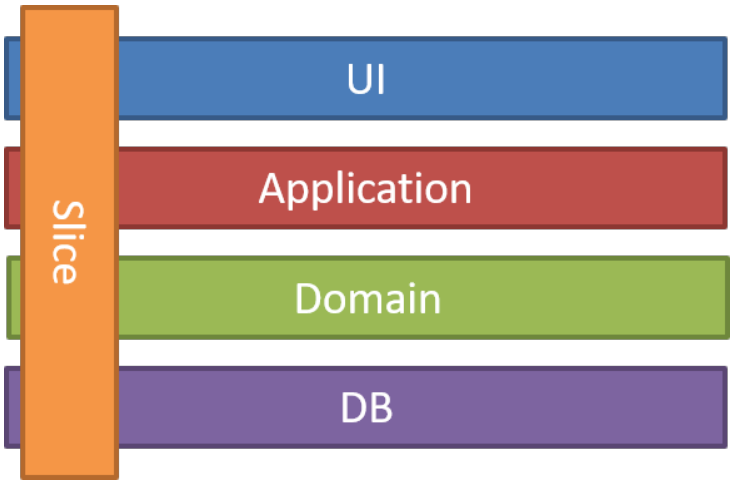


Abbildung 11: Beispiel einer Vertical Slice Architecture. Übernommen aus [5]

Die Features von Securaware können sich ändern und sind nicht fixiert, wie zum Beispiel die Features einer Bank. Diese Ungewissheit spricht gegen Feature-unterteilende Architekturen. Zusätzlich teilen viele Features technischen Concerns, wie zum Beispiel die Authentifizierung. Entsprechend werden die zwei Feature-unterteilende Architekturen Vertical Slice und Clean Architecture nicht verwendet. Das Phishing Backend kommuniziert mit drei externen Systemen. Der Fokus des Backends liegt jedoch nicht auf diesen Schnittstellen, sondern auf dem Backend an sich. Folglich findet die Hexagonal Architecture keine Anwendung. Entsprechend kommen die Layered und Onion Architecture infrage. Die Layered und Onion Architecture unterscheiden sich nicht siginifikant von einander. Ein Unterschied jedoch ist, dass die Onion Architecture auf eine strikte Trennung der Kernlogik von Schnittstellen zu externen Systemen zielt. Zusätzlich setzt die Onion Architecture auf

das Dependency Inversion Principle. Je nach Quelle verwendet auch die Layered Architecture dieses Prinzip, die Grenzen dieser Architekturen sind fliessend. Folglich wird das Phishing Backend gemäss der Onion Architecture strukturiert.

Es werden bewusst wenige Konventionen der Onion Architecture nicht eingehalten. Diese werden aufgelistet:

- **ORM und domain models**: Das verwendete ORM wird über Go tags in den domain models konfiguriert. Es werden keine dedizierten data persistence primitives verwendet, um Zeit und Code zu sparen. Folglich hängt der Domain-Ring von der externen ORM - Bibliothek ab. Diese Abhängigkeit wird akzeptiert.
- **Go „internal"-Ordner**: Gemäss Onion Architecture soll der Top-Level-Ordner in Ringe aufgeteilt werden. Aufgrund von Go-Eigenschaften wird die Onion Architecture im Ordner „internal" angewendet: Go-Dateien im „internal"-Ordner können nicht von externen Projekten verwendet werden. Entsprechend wird fast der gesamte Go-Code in diesem Ordner platziert und strukturiert.
- **Automatisierte Testfälle**: Gemäss Theorie sind die automatisierten Testfälle Teil des äussersten Ringes. Die Entwickler von Go empfehlen, Unit-Tests im selben Ordner wie den Production-Code zu platzieren [38]. Gemäss eigener Erfahrung halten sich fast alle Go-Projekte an diese Konvention. Entsprechend werden Unit-Tests gemäss der Konvention der Go-Entwickler und nicht gemäss der Onion Architecture platziert.
- **Äusserster Ring**: Der Äusserste Ring wird als „adapters" bezeichnet. Grund dafür ist, dass der name „adapters" sprechender ist für das Projekt.

5.3.2. Ordnerstruktur

Die Ordnerstruktur wird gemäss der Onion Architecture angeordnet. Abbildung 12 visualisiert diese Ordnerstruktur.

Abbildung 12: Ordnerstruktur gemäss Onion Architecture des Phishing Backends

Die in Abbildung 12 präsentierte Struktur wird genauer beschrieben:

- .docker: Docker-bezogene Konfigurationen
- cmd: Ausführbare Applikationen. In diesem Fall eine, das Phishing Backend.
- integration_tests: automatisierte Integration Tests und dessen Setup
- internal
  - adapters
    - communication: E-Mail-Versandt bezogene Implementation
    - persistence: Datenbank bezogene Implementation
    - presentation: HTTP bezogene Implementation: Controllers, HTTP-Error-Handling, generierter HTTP-DTO-Code und Mappers
  - domain_model
  - domain_services
    - interfaces: Datenbank- und E-Mail-Schnittstellen , welche von den Adapters implementiert werden
    - services: Businessfunktionalität

5.3.3. Externe Softwarebibliotheken

Bibliothek	**Verwendung**
github.com/golang-jwt/jwt/v5	Erstellen und validieren von JSON Web Token (JWT)
github.com/google/uuid	Erstellen von UUIDv4
github.com/jackc/pgx/v5	Toolkit für PostgreSQL-Verbindungen und Fehlerbehandlung
github.com/jackc/tern/v2	Datenbankmigrator. Die Bibliothek wird weiter im Abschnitt 5.6.2 beschrieben.
github.com/oapi-codegen/runtime	Notwendige Runtime-Bibliothek für generierten Code aus der OpenAPI-Definition.
gorm.io/gorm	ORM für Go
gorm.io/driver/postgres	Datenbanktreiber für PostgreSQL
github.com/stretchr/testify	Toolkit mit unter anderem Assertions für automatisierte Testfälle
go.uber.org/mock	Toolkit zum erstellen von Mocks für Unit Testfälle

Tabelle 25: Externe Softwarebibliotheken des Phishing Backends

5.3.4. Connectivity zum Phishing Forwarder

E-Mails müssen sicher dem Phishing Forwader übergeben werden, sodass Vermittler die Phishind-Simulations-E-Mails nicht als Spam deklarieren. Für diese Security werden Standards und Protokolle aus dem Abschnitt 2.1.4.3 verwendet.

Es wird ESMPT als Basis verwendet. Für die Verschlüsselung der Kommunikation wird STARTTLS verwendet. Für die Authentifizierung wird SMPT-Auth verwendet.

5.3.5. HTTP-Model-Generierung

Die Software-Bibliothek github.com/oapi-codegen/oapi-codegen generiert anhand einer OpenAPI-Spezifikation Go-HTTP-Client und oder -Server-Code [39]. Der generierte Code ist der gegebenen Schnittstelle konform. Das Phishing Backend lässt sich Go-Server-Code generieren. Der Code wird zur Compile-Zeit generiert und ist in Git eingebunden. Bis auf externe Bibliothek, welche generierte Typen deklariert, sind keine Laufzeitnachteile vorhanden. Entsprechend wird diese Bibliothek verwendet, da die Vorteile die Nachteile überwiegen. Der Code kann über die Go-Direktive generate und die externe Bibliothek als Tool erstellt werden. Code 3 zeigt dies auf.

```
1 package api
2
3 // https://github.com/oapi-codegen/oapi-codegen/
4 //go:generate go tool oapi-codegen -config server-side-code-generation-
  config.yaml /api/phishing-backend-open-api.yaml
```

Code 3: Simplifizierte Go-HTTP-Server-Code-Generierung

5.3.6. Implementierung der Online-Kurs-Tests

Die Schnittstelle zum Tests abzuschliessen, erwartet alle gegebenen Antworten des Benutzers. Das Phishing Backend berechnet anschliessend die Punktzahl indem die Anzahl falscher Antworten berechnet wird.

Die Berechnung dieser falscher Antworten kann auf die Mengenlehre reduziert werden:

$$
\begin{aligned}
A &= \text{Menge der Antworten des Benutzers} \\
B &= \text{Menge der möglichen Antworten} \\
C &= \text{Menge der korrekten Antworten} \\
C &\subset B
\end{aligned}
$$

$$
\begin{aligned}
\text{Anzahl falscher Antworten} &= \text{Inkorrekte Antworten} + \text{Fehlende korrekte Antworten} \\
&= |A \setminus C| + |C \setminus A| \\
&= |(A \setminus C) \cup (C \setminus A)| \quad (1) \\
&= |A \Delta C|
\end{aligned}
$$

$$
(1) \text{ Diese Transofrmation ist erlaubt, da } (A \setminus C) \cap (C \setminus A) = \emptyset
$$

Die Anzahl falscher Antworten kann folglich über symmetrische Differenz oder Kontravalenz der Menge der Antworten des Benutzers und der Menge der korrekten Antworten ausgerechnet werden. Code 4 implementiert diese symmetrische Differenz.

```go
type userAndActualAnswer struct {
  totalAnswers   int
  correctAnswers utils.Set[uuid.UUID]
  userAnswers    utils.Set[uuid.UUID]
}

// A: set of answers the user thinks are correct
// B: set of all actual correct answers
// ex: A={1,4,5}, B={1,3} -> |(A - B) U (B - A)| = |{4,5} U {3}| = |{3,4,5}| = 3
func (u *userAndActualAnswer) getNumberOfWrongOrMissingAnswers() int {
  wrong := 0
  wrong += u.correctAnswers.Difference(&u.userAnswers).Size()
  wrong += u.userAnswers.Difference(&u.correctAnswers).Size()
  return wrong
}
```

Code 4: Berechnung der Anzahl falschen oder fehlenden Antworten mit der Kontravalenz

5.4. Phishing Forwarder

Der Phishing Forwarder dient als Mail Transfer Agent (MTA) . Dieser MTA soll über standardisierte E-Mail-Protokolle wie zum Beispiel ESMPT angesprochen werden. Dies bringt den Vorteil, dass dieser MTA sehr einfach ausgetauscht werden kann oder ohne grossen Aufwand weitere MTAs hinzugefügt werden können.

Auf die Implementation eines eigenen MTA wird verzichtet, da bereits viele open source Implementationen existieren. Es wurde sich für das Projekt Postfix [40] entschieden.

Damit die Kommunikation mit STARTTLS verschlüsselt verwendet werden kann, ist ein signiertes Zertifikat einer certificate authority notwendig. Das Zertifikat wird mithilfe des Tools Certbot [41] erstellt, welches das Zertifikat von Let's Encrypt [42] bezieht. Let's Encrypt ist eine nonprofit certificate authority, welche kostenfrei Zertifikate ausstellt. Certbot ist ein open source Tool, welches den gesamten Prozess von der Schlüsselgenerierung bis hin zum Anfordern des Zertifikats automatisiert. Certbot wird so konfiguriert, sodass die Zertifikate automatisch erneuert werden.

5.5. Securaware Proxy

Als Proxy wird die etablierte und aus dem Studium bekannte Software traefik eingesetzt [43]. Traefik-Konfigurationen werden in statische und dynamische Konfigurationen unterteilt. Die dynamischen Konfigurationen sind dynamisch, weil diese sich der Umgebung anpassen, sei es dem Datenverkehr oder der Docker-Umgebung. Die statischen Konfigurationen werden in Code 5 beschrieben, die Dynamischen in Code 6.

In Fall von Securaware sind hauptsächlich fünf Konfigurationen notwendig: Entrypoints, Middlewares, Routers, Services und Certificate Resolvers. Die Interaktion wird in Abbildung 13 veranschaulicht. Entrypoints definieren, über welche eingehenden Verbindungen, Netzwerkpakete von Traefik akzeptiert werden sollen. Services definieren,

wohin die Netzwerkpakete weitergeleitet werden. Routers verbinden Entrypoints mit Services, sie definieren, welche Netzwerkpakete zu welchem Service weitergeleitet werden. Middlewares können in Routern verwendet werden, um eingehende Anfragen vor der Weiterleitung zu einem Service zu modifizieren, z.B. durch Weiterleitungen, Authentifizierung oder Header-Anpassungen. Mittels Certificate Resolvers lassen sich zudem einem Router TLS-Zertifikate zuweisen, die der Router ausliefern soll.

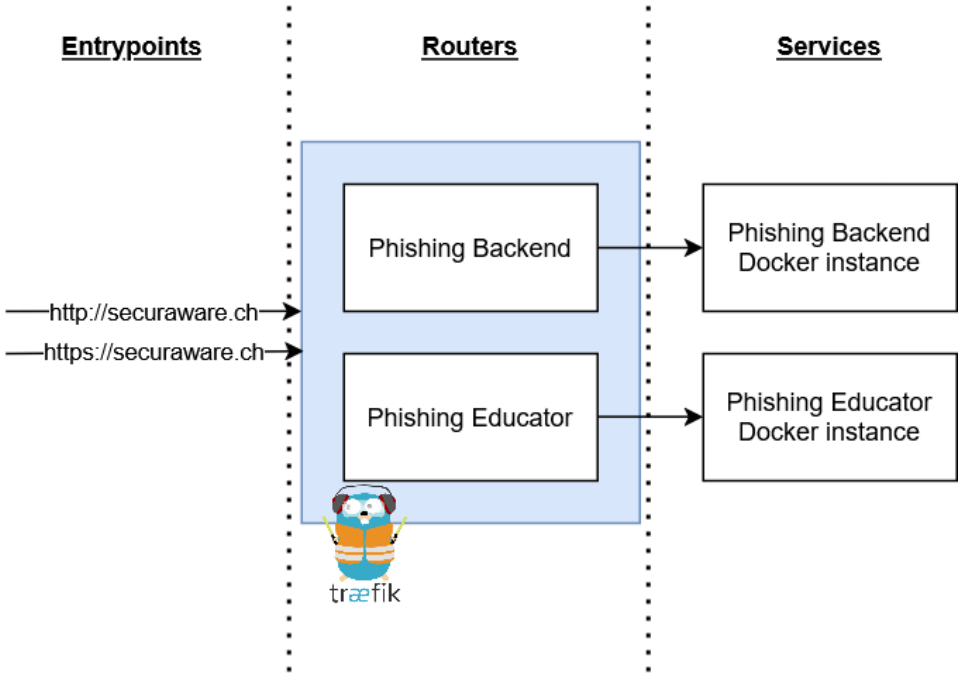


Abbildung 13: Simplifizierte Traefik-Konfigurationen im Überblick. Eigene Darstellung unter Verwendung des Traefiklogos von [6]

5.5.1. Statische Konfigurationen

```yaml
entryPoints:
  web:
    address: ":80"
    http:
      redirections:
        entryPoint:
          to: websecure
          scheme: https
          permanent: true

  websecure:
    address: ":443"

certificatesResolvers:
  myresolver:
    acme:
      email: mischa.binder@stafag.ch
      storage: acme.json
      tlsChallenge: {}

providers:
  file:
    filename: "/etc/traefik/traefik_dynamic.yaml"
```

Code 5: Simplifizierte statische Traefik-Konfigurationen (traefik.yaml)

Beschreibung zu den statischen Konfigurationen aus Code 5:

- entryPoints: Securaware akzeptiert sowohl HTTP als auch HTTPS. HTTP-Anfragen werden zu HTTPS-Anfragen weitergeleitet, sodass diese auch verschlüsselt sind.
- certificatesResolvers: Konfiguriert Automatic Certificate Management Environment (ACME) . Dies sorgt dafür, dass die public und private Keys für TLS und dessen Zertifikat automatisch erneuert werden.
- providers: Gibt den Provider für die dynamischen Konfigurationen an.

5.5.2. Dynamischen Konfigurationen

In Securawares Fall ist die Dockerumgebung statisch und der Datenverkehr überschaubar. Entsprechend wird auf Skalierungsmöglichkeiten verzichtet.

```yaml
http:
  routers:
    phishingBackend:
      rule: "(Host(`www.securaware.ch`) || Host(`securaware.ch`)) &&
  (Path(`/api`) || PathPrefix(`/api/`))"
      service: phishingBackend
      entrypoints:
        - websecure
      middlewares:
        - redirect-root-to-www
      priority: 10

    phishingEducator:
      rule: "Host(`www.securaware.ch`) || Host(`securaware.ch`)"
      service: phishingEducator
      entrypoints:
        - websecure
      middlewares:
        - redirect-root-to-www
      priority: 1

    phishingDomains:
      rule: "!Host(`www.securaware.ch`) && !Host(`securaware.ch`)"
      entrypoints:
        - web
        - websecure
      middlewares:
        - redirect-phishing-domains-to-securaware

  services:
    phishingBackend:
      loadBalancer:
        servers:
          - url: "http://phishing_backend:8080"

    phishingEducator:
      loadBalancer:
        servers:
          - url: "http://phishing_educator:80"
```

Code 6: Simplifizierte dynamische Traefik-Konfigurationen (traefik_dynamic.yaml)

Beschreibung zu den dynamischen Konfigurationen aus Code 6:

- Zeilen 3 - 10: Anfragen vom Frontend zum Backend werden über diesen Router verwaltet. Hierfür muss die Securaware-Domain übereinstimmen und „api“ im Pfad beinhaltet sein. Die Domain muss wegen der Phishing Simulation übereinstimmen, welche später beschrieben wird. Dieser Router wird mit einer höheren Priorität ausgeführt, weil der Educator und das Backend dieselbe Domain verwenden.

- Zeilen 12 - 19: Alle übrigen Anfragen zu Securaware werden zum Educator weitergeleitet.
- Zeilen 21 - 27: Für die Phishing-Simulation wird ein spezieller Router eingesetzt. Die in der Simulation verwendeten Links verweisen auf verschiedene Domains, beispielsweise „bank-of-switzerland“. Wenn ein Benutzer auf einen dieser bösartigen Links klickt und dadurch nicht erkennt, dass es sich um eine Phishing-Mail handelt, erfolgt eine Weiterleitung zu Securaware. Dort werden die relevanten Merkmale aufgezeigt, anhand derer die E-Mail als Spam bzw. Phishing-Versuch hätte erkannt werden können. Der Router übernimmt die Entgegennahme der Anfragen und leitet diese an Securaware weiter. Über die Query-Parameter werden zusätzlich Benutzerdaten übermittelt, um nachvollziehen zu können, welcher Nutzer auf einen bösartigen Link der Phishing-Simulation geklickt hat.

5.6. Phishing Mail Storage

Dieses Unterkapitel beschreibt, in welchen Entitäten Daten gespeichert werden und wie Datenbankänderungen umgesetzt werden.

5.6.1. Entitäten

Die Entitäten werden in Abbildung 14 in Form eines Entity-Relationship Diagram (ERD) und einer anschliessenden Beschreibung beschrieben.

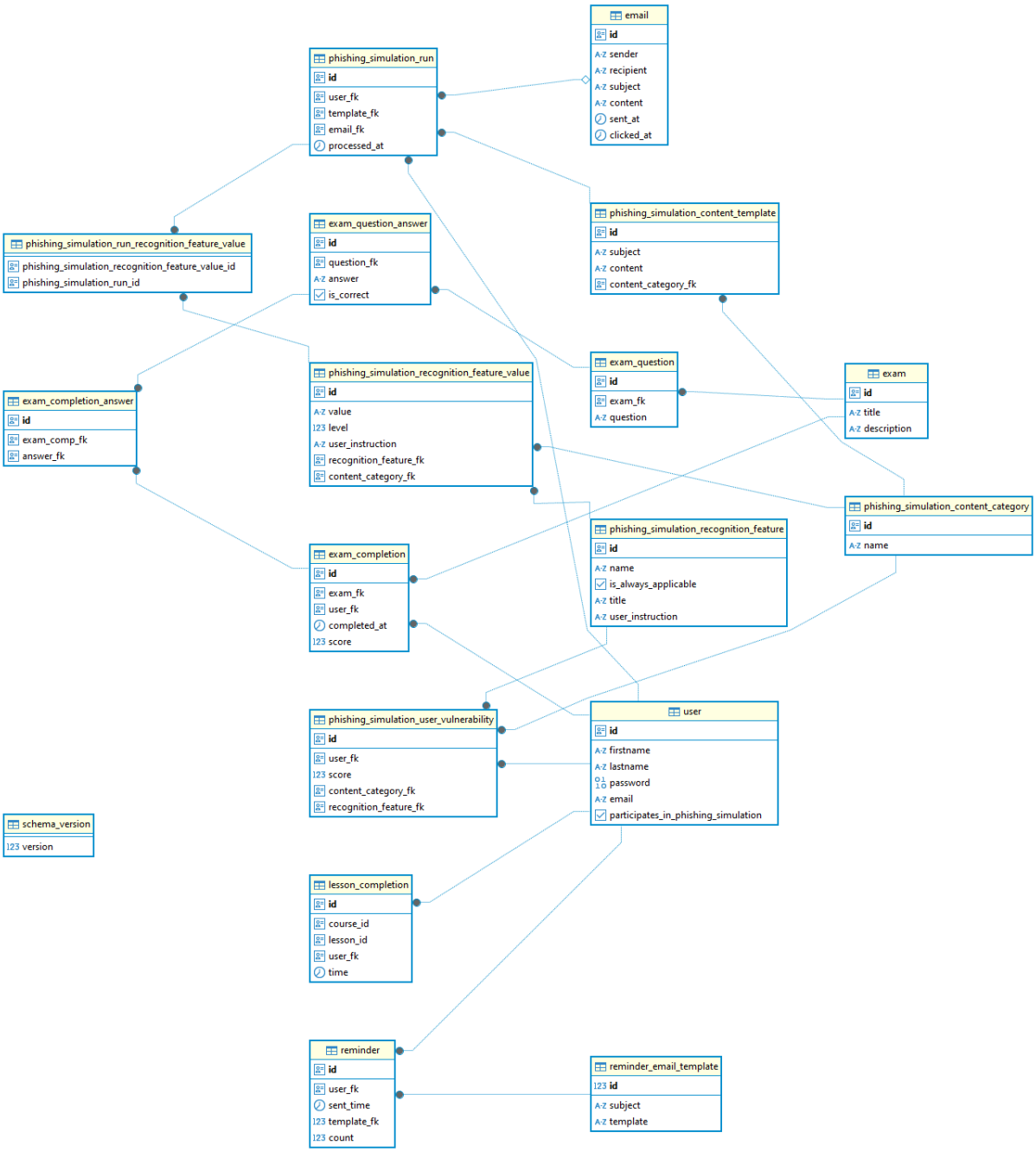


Abbildung 14: Entity-Relationship Diagram von Securaware

Nachfolgend werden die Entities in Kategorien unterteilt und beschrieben.

Benutzer

- User: Benutzerdaten wie zum Beispiel E-Mail oder Vorname

Lektionen

- Lesson Completion: Abgeschlossene Lektionen eines Benutzers

Tests

- Exam: Der Test an sich
- Exam Question: Frage in einem Test

- Exam Question Answer: Antwort zu einer Frage in einem Test
- Exam Completion: Vollendeter Test eines Benutzers
- Exam Completion Answer: Abgegebene Antwort eines Benutzers in einem vollendeten Test

Phishing Simulation
Aus Übersichtswecken wird nachfolgend Phishing Simulation durch […] ersetzt.
- […] Run: Ausführung einer Phishing Simulation, nämlich eine Spam-E-Mail
- […] User Vulnerability: Zusammenfassung, wie gut der Benutzer Spam-E-Mail erkennen kann und welche Merkmale dieser nicht gut erkennen kann.
- […] Content Category: Art des Angriffs
- […] Recognition Feature: Merkmal, wie der Angriff durchschaut werden kann
- […] Run Recognition Feature Value: Beziehungsentität, sodass ein Run mehrere Merkmale aufweisen kann
- […] Recognition Feature Value: Wert, welcher Platzhalter in der E-Mail-Vorlage ersetzt
- […] Content Template: E-Mail-Vorlage
- Email: Effektiv versendetes E-Mail mit Text. Wird für die Benutzeroberfläche verwendet.

Erinnerung
- Reminder: Versendete Erinnerung
- Reminder Email Template: E-Mail-Vorlage für eine Erinnerung

Technisch
- Schema Version: Wird vom Schemamigrator verwendet, um die letzte angewendete Änderung zu speichern

5.6.2. Schemamigration
Wenn Securaware produktiv eingesetzt wird, werden reale Kundendaten in der Datenbank persistiert. Diese dürfen nicht verloren gehen. Daher müssen Änderungen am Datenbankschema so umgesetzt werden, dass sie mit dem bestehenden Schema kompatibel sind. Ein vollständiger Austausch des Schemas ist nicht zulässig, da dies unweigerlich zu Datenverlust führen würde.

Zudem ist es essenziell, fehlerhafte Änderungen so zu behandeln, dass die Datenbank stets in einem konsistenten und betriebsbereiten Zustand verbleibt. Schemaänderungen dürfen entsprechend auch nur ein Mal angewendet werden. Diese Anforderungen werden durch den Einsatz eines Datenbankschema-Migrationssystems erfüllt.

Als Migrationswerkzeug kommt Tern zum Einsatz, siehe Tabelle 25. Beim Start des Phishing-Backends prüft Tern, welche Migrationen bereits angewendet wurden, und führt ausstehende Migrationen automatisch aus. Hierzu verwaltet Tern eine eigene Tabelle, in welcher die letzte angewendete Änderung notiert wird. Um das Schema anzupassen, wird eine neue SQL-Datei mit den erforderlichen SQL-Statements erstellt und in das vorgesehene

Verzeichnis im Phishing Backend verschoben. Nach einem Neustart des Phishing-Backends werden die Änderungen übernommen.

Durch dieses Vorgehen ist sichergestellt, dass das ORM -Änderungen im Phishing Backend und das Datenbankschema stets synchron sind.

5.7. Docker

Docker wird verwendet, um die Anforderung „NFR4 Portability“ gemäss Tabelle 4 umzusetzen. Bis auf den Phishing Forwarder sind die Docker- und C4-Container identisch, die C4-Container werden als jeweils einen Docker-Container realisiert.

5.7.1. Container-Orchestrierung

Die Docker-Container sind voneinander abhängig, kommunizieren untereinander und müssen entsprechend als ganzes funktionieren und verwaltet werden. Entsprechend wird eine Container-Orchestrierungsstrategie benötigt. Docker stellt mit Docker Compose eine automatisierte Softwarelösung bereit. Eine Alternativlösung stellt Kubernetes dar. Kubernetes weist mächtigere Funktionen auf und bietet Skalierungsmöglichkeiten. Jedoch ist das Aufsetzen und Warten von Kubernetes komplexer als Docker Compose. Da es sich bei Securaware um einen Prototyp handelt und bisher keine praktischen Erfahrungen im Umgang mit Kubernetes vorliegen, wird Docker Compose verwendet. Die Nutzung von Kubernetes wird verworfen und nicht weiter verfolgt.

Mittels Docker Compose lassen sich zudem eigenständige Compose Files erstellen, um diese Orchestrierung sowohl in der Entwicklungsumgebung als auch der Produktionsumgebung verwenden zu können.

Aus Sicherheitsgründen sollten Zugangsdaten der Produktionsumgebung nicht in die Versionsverwaltung aufgenommen werden. Aus diesem Grund werden sämtliche Umgebungsvariablen inkl. Zugangsdaten direkt auf dem Produktionsserver abgelegt und nicht über CI/CD aus der Versionsverwaltung ausgeliefert.

Im Docker Compose File der Produktionsumgebung wird für Services mit eigens erstellten Images eine Image-URL der GitLab CI-Registry hinterlegt. Dadurch lässt sich direkt mit Docker Compose ein Image erzeugen und in die Registry speichern. Damit dieser Prozess allerdings ohne die Präsenz der Umgebungsvariablen, sondern lediglich jene, die für den Build-Prozess benötigt werden, funktioniert, wird das Compose File der Produktionsumgebung in zwei Compose Files aufgeteilt. Nur eines der beiden wird für den Build-Prozess sowie das Speichern in der Registry verwendet. Zur Auslieferung der Software in die Produktionsumgebung werden beide Files verwendet, wobei das zweite File das erste um die Definition der zusätzlichen Umgebungsvariablen resp. Referenzen auf .env-Files ergänzt.

5.7.2. Netzwerk

Die Docker-Container sind deckungsgleich. Entsprechend kommunizieren die Docker-Container untereinander, wie beschrieben in Abschnitt 4.3.1.2. Für diese Kommunikation sind Netzwerke notwendig. Gemäss offizieller Dokumentation sind benutzerdefinierte Netzwerke empfohlen „Use user-defined bridge networks [...]. This is recommended for

standalone containers running in production.“ [44]. Es wird gleichzeitig davon abgeraten, das Standardnetzwerk zu verwenden „Remember, the default bridge network is not recommended for production.“ [44]. Entsprechend werden benutzerdefinierte Docker Bridge Networks verwendet.

Pro erlaubter Kommunikation zwischen zwei Containern wird ein Bridge Network erstellt. In diesem Netzwerk sind jeweils nur die zwei erlaubten Container und keine weiteren. Dadurch sind nur Kommunikationen gemäss Abschnitt 4.3.1.2 möglich.

In Docker compose wird dies wie folgt umgesetzt.

```yaml
services:
  phishing_backend:
    networks:
      - frontend___backend
      - backend___db

  phishing_frontend:
    networks:
      - frontend___backend

  phishing_mail_storage:
    networks:
      - backend___db

networks:
  backend___db:
    driver: bridge
  frontend___backend:
    driver: bridge
```

Code 7: Simplifiziertes und auf Docker-Netzwerk reduziertes compose.yaml

5.7.3. Phishing Backend

In diesem Unterkapitel wird das Dockerfile des Phishing Backend beschrieben. Dieses ist in mehrere Docker stages aufgeteilt [45], verwendet go caches [46], Docker cache mounts [47] und einen Nichtroot-Benutzer zum Ausführen der Applikation. Code 8 beschreibt das Dockerfile.

```
1  FROM golang:1.24.1 AS build
2  WORKDIR /app
3
4  ENV GOCACHE=/go-cache
5  ENV GOMODCACHE=/gomod-cache
6
7  COPY go.mod go.sum ./
8  RUN --mount=type=cache,target=/gomod-cache go mod download
9  COPY . .
10
11 RUN --mount=type=cache,target=/gomod-cache --mount=type=cache,target=/
   go-cache CGO_ENABLED=0 go build -o /bin/phishingBackend cmd/main.go
12
13
14 FROM alpine:3.21.3
15 COPY --from=build /bin/phishingBackend /app/phishingBackend
16
17 RUN adduser -D nonrootuser && chown -R nonrootuser:nonrootuser /app
18 USER nonrootuser
19 CMD ["/app/phishingBackend"]
```

Code 8: Dockerfile des Phishing Backend

- Zeile 1: Das offizielle Dockerimage von Google wird als Basis verwendet, damit Go-spezifische Tools wie zum Beispiel der Compiler benutzt werden können.
- Zeilen 4 & 5: Es werden zwei Go-spezifische Umgebungsvariablen gesetzt, welche den Pfad für Build Caches angeben [46]. Diese Caches beschleunigen das Herunterladen von externen Abhängigkeiten und das Kompilieren der Applikation.
- Zeile 7: Zum Beschleunigen des Erstellens des Docker-images, werden externe Bibliotheken nur heruntergeladen, falls diese sich verändert haben. Dies wird bewerkstelligt, in dem in einem ersten Schritt die zwei Dateien, welche externe Bibliotheken auflisten (go.mod und go.sum), kopiert werden und in einem zweiten Schritt diese Bibliotheken mit Cacheoptimierung heruntergeladen werden. Aufgrund der Dockerschichten werden diese zwei Schritte nur ausgeführt, wenn sich die zwei genannten Dateien verändern, verglichen zum letzten Build.
- Zeile 11: Das Phishing Backend wird in diesem Schritt kompiliert, wobei die in Zeilen 4 & 5 beschriebenen Caches verwendet werden: Beim Erstellen des Binary entstehen Artefakte. Diese werden gemäss den Pfaden der Umgebungsvariablen abgelegt, sodass diese in zukünftigen Builds verwendet werden können. Aufgrund des Schichtenmodells von Docker werden Docker cache mounts [47] verwendet. Cache mounts bieten eine Möglichkeit, einen persistenten Speicherort für den Cache anzugeben, welcher keine Schichten invalidiert und über mehrere Builds hinweg verwendet werden kann. Neben den cache mounts wird die Umgebungsvariable „CGO_ENABLED“ gesetzt. Diese gibt an, ob C-Code linked werden soll oder nicht. Um das Runtime-Docker-Image simple zu halten, wird auf C-Code verzichtet. Entsprechend ist das resultierende Binary statisch und weist keine Abhängigkeiten zu externem C-Code auf.

- Zeile 14: Ein weiteres Basisimage wird angegeben, womit ein Multi-stage build [45] realisiert wird. Es wird alpine als Basisimage verwendet, da dies relativ kompakt ist und gleichzeitig essenzielle Dateien für die Runtime beinhaltet wie zum Beispiel die Liste der akzeptierten Root Certificate Authorities, welche für Transport Layer Security (TLS) benötigt werden. Entsprechend wird auf ein distroless Image verzichtet.
- Zeilen 17 & 18: Aus Sicherheitsgründen wird ein Benutzer und eine Gruppe ohne Rootrechte erstellt. Diese haben Zugriff auf den Ordner, in welchem das Backendbinary sich befindet. Dieser Benutzer wird verwendet, um die Applikation zu starten. Falls durch einen Exploit ein Angreifer Zugriff auf den Container erlangt, hat dieser keine Rootrechte, weil dieser Benutzer verwendet wird.

5.7.4. Phishing Educator

Code 9 beschreibt das Dockerfile des Phishing Educators. Auch dieses ist in mehrere Docker stages aufgeteilt.

```
1  FROM node:22-alpine AS build
2  WORKDIR /app
3
4  COPY package.json package-lock.json ./
5  RUN npm ci
6
7  COPY . .
8  COPY .docker/phishing_educator.env .env
9
10 RUN npm run build
11
12 FROM nginx:alpine
13 COPY --from=build /app/build/client /usr/share/nginx/html
14
15 COPY .docker/nginx.conf /etc/nginx/conf.d/default.conf
16
17 EXPOSE 80
18
19 CMD ["nginx", "-g", "daemon off;"]
```

Code 9: Simplifiziertes Dockerfile des Phishing Educators

- Zeile 4 und 5: Externe Bibliotheken werden durch diese Konfigurationen nur heruntergeladen, falls neue Bibliotheken eingebunden oder bestehende entfernt werden.
- Zeile 10: Erstellt den Phishing Educator in statisch auslieferbare Dateien.
- Zeile 12: Benutzt nginx, um die nötigen statischen Dateien Phishing Educator dem Benutzer zu übergeben.
- Zeile 19: Starte nginx als Vordergrundprozess und nicht als Hintergrundprozess (daemon). Neben nginx soll kein weiterer Prozess in dem entstehenden Docker-Container ausgeführt werden.

5.8. Softwarecode-Testkonzept

Um die Korrektheit von Securaware zu gewährleisten, werden, wie in der Praxis üblich, automatisierte Testfälle geschrieben. Diese können in drei Kategorien eingeteilt werden; Unit-, Integration- und End-to-End-Testfälle. Diese unterscheiden sich unter anderem im Scope, beziehungsweise der zu verifizierenden Qualität, dem Implementationsaufwand und dem Testsetup.

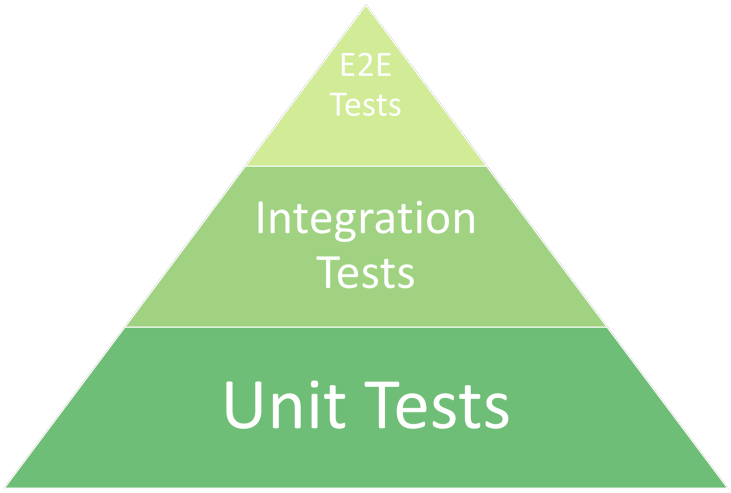


Abbildung 15: Testingpyramide. Die verschiedenen Arten von automatisierten Testfällen werden in Form einer Pyramide aufgelistet.

Die Abbildung 15 und die verschiedenen Testkategorien werden in der Tabelle 26 genauer erläutert.

Testkategorie	Beschreibung
Unit Tests	Die Unit Tests stellen das Fundament der automatisierten Testfälle dar (siehe Abbildung 15). Diese weisen den geringsten Implementationsaufwand auf. Sie verifizieren, dass ein einzelnes Codefragment funktioniert. Solch ein Codefragment kann eine Klasse, eine Funktion, eine Invarianz einer Klasse oder auch das Zusammenspiel mehrerer Funktionen sein. Gemäss C4-Theorie werden die Interaktionen auf Class-Level verifiziert. Das Testsetup beansprucht, dass der zu testende Code nicht von Services abhängig ist. Entsprechend müssen Abhängigkeiten mit einer dedizierten Testklasse ausgetauscht werden. Diese Testklasse gibt Testwerte zurück oder weist ein angepasstes Verhalten auf. In der Norm stellen diese Tests die Mehrheit der automatisierten Testfälle dar.
Integration Tests	Integration Tests verifizieren das Zusammenspiel mehrerer Codefragmente. Gemäss C4-Theorie wird die Interaktion zwischen Komponenten verifiziert. Container werden im Test-setup vorgetäuscht. Container vorzutäuschen ist aufwändiger zu implementieren und Tests benötigen mehr Durchlaufzeit, entsprechend gibt es in der Norm eine grössere Anzahl Unit als Integration Tests.
End-to-End Tests	End-to-End Tests verifizieren das Zusammenspiel mehrerer Softwaresysteme. Hierfür sind Testumgebungen mit dedizierten Servern notwendig, sodass die Produktionsserver und -datencenter nicht tangiert werden. Das Setup ist aufwändig und kostenintensiv, widerspiegelt aber die Benutzerinteraktionen am genauesten.

Tabelle 26: Beschreibung der verschiedenen Arten von Testfällen

Auf automatisierte End-to-End-Tests wird im Falle von Securaware bewusst verzichtet. Da die Frontend-Komponenten der Applikation eine geringe Komplexität aufweisen, lassen sich solche Testdurchläufe mit vergleichsweise geringem Aufwand manuell durchführen. Der für die Implementierung solcher automatisierten Tests benötigte Aufwand rechtfertigt aus diesem Grund dessen Ertrag in diesem Fall nicht. Infolgedessen werden End-to-End-Tests sowie die gesamten Tests der Frontend-Komponenten manuell durchgeführt.

Die Testfälle werden gemäss Behavior-Driven Development (BDD) erstellt und strukturiert [48]. Dabei werden Tests in drei Phasen aufgeteilt:
1. Vorbereitung (given): Testdaten für den zu testenden Code werden erstellt. Prerequisites des Codes werden erfüllt.
2. Ausführung (when): Der zu testenden Code wird ausgeführt.
3. Verifikation (then): Die Resultate des zu testenden Codes werden auf ihre Korrektheit überprüft.

Die Integration Tests des Phishing Backends replizieren das Verhalten des Benutzers: Daten und Operationen werden ausschliesslich über die HTTP-Schnittstelle generiert beziehungsweise ausgeführt. Darüber hinaus setzen die Tests eine PostgreSQL-

Datenbankinstanz voraus, die vom zu testenden Code verwendet wird. Auf diese Weise wird die Datenbankinteraktion ebenfalls getestet. Abgesehen von der Verifikationslogik wird kein Production-Code direkt aufgerufen. Diese realitätsnahe Testumgebung gewährleistet funktionale Korrektheit.

5.9. Usability-Test

Im Rahmen der Entwicklung der Plattform fliesst ein Usability-Test in den Prozess ein, um möglichst früh Rückmeldungen zur Nutzerführung und zum Interaktionsdesign zu erhalten. Dabei stehen Aspekte wie die Auffindbarkeit zentraler Funktionen, die Verständlichkeit der Benutzeroberfläche sowie die visuelle Rückmeldung innerhalb des Systems im Vordergrund. Auch der Umgang mit Lernfortschritt, Kursstruktur und Einstellungen wird genauer betrachtet, um potenzielle Hürden in der Nutzung frühzeitig zu erkennen.

5.9.1. Testaufbau und Methodik

Der Usability-Test erfolgt als qualitativer explorativer Test unter Verwendung des Thinking-Aloud-Verfahrens [49]. Die Testperson erhält dabei mehrere Aufgaben und beschreibt während der Bearbeitung ihre Gedanken. Es wird beobachtet, an welchen Stellen Unsicherheiten auftreten, welche Erwartungen an die Benutzeroberfläche bestehen und welche Interaktionen intuitiv erfolgen.

Die Aufgaben orientieren sich dabei an realitätsnahen Nutzungsszenarien der Plattform. Ziel ist es, ein tiefes Verständnis der Wahrnehmung und Nutzung zu erhalten, nicht jedoch die Leistungsfähigkeit oder Korrektheit der Antworten zu bewerten.

5.9.2. Durchführung

Der Usability-Test umfasst drei zentrale Nutzungsszenarien, die durch Testpersonen bearbeitet werden: die Registrierung & Informationssuche zum Thema Phishing via E-Mail, die Orientierung im Level- und Punktesystem sowie die Anmeldung zur automatisierten Phishing-Simulation. Die Szenarien orientieren sich an realitätsnahen Nutzungskontexten und sollen typische Interaktionen mit der Plattform abbilden. Sämtliche Beobachtungen und Diskussionen während des Usability-Tests werden in einem Protokoll festgehalten. Jenes Protokoll kann dem Anhang entnommen werden.

Während der Durchführung werden verschiedene Aspekte der Benutzerfreundlichkeit beobachtet. Zentrale Funktionen können dabei grundsätzlich ohne grössere Schwierigkeiten gefunden und genutzt werden. Vereinzelte Irritationen ergeben sich durch inhaltliche Überschneidungen innerhalb der Kursstruktur sowie durch uneinheitliche Gestaltung vereinzelter Interface-Elemente. Die Logik des Fortschrittssystems wird als nicht vollständig nachvollziehbar wahrgenommen. Positiv fällt die intuitive Navigation auf, insbesondere bei der Nutzung etablierter UI-Konventionen.

Zusätzlich wird angemerkt, dass der Ablauf der Prüfungsfunktion zu Beginn nicht ausreichend erläutert wird. Es wird vorgeschlagen, eine kurze Einführung zum Prüfungsprozess bereitzustellen, um die Verständlichkeit und Akzeptanz dieser Funktionalität zu verbessern.

5.9.3. Auswertung und Erkenntnisse

Der Usability-Test liefert mehrere aufschlussreiche Erkenntnisse:

- Intuitive Navigation: Die Orientierung über den Avatar funktioniert erwartungsgemäss gut, was auf eine konsistente Nutzung bekannter UX -Muster hinweist.
- Verbesserungspotenzial der Kursstruktur: In der aktuellen Kursübersicht erschwert die thematische Überschneidung einzelner Kurse die gezielte Auswahl passender Inhalte. Da Kurse lediglich mit einem Titel und einer kurzen Beschreibung dargestellt werden, ist für Nutzer:innen nicht sofort ersichtlich, worin sich die Kurse unterscheiden. Um die Orientierung zu verbessern, sollten inhaltliche Abgrenzungen klarer gestaltet oder Überschneidungen explizit als Teil des didaktischen Konzepts gekennzeichnet werden.
- Feature-Findability: Funktionen wie die Einstellung für die Anmeldung zur Phishing-Simulation sind gut auffindbar, was auf eine gute Informationsarchitektur in den Einstellungen hinweist.
- Onboarding: Die Prüfungsfunktion benötigt eine Einführung, z.B. durch ein erklärendes Popup beim ersten Aufruf.

5.9.4. Fazit

Der Usability-Test zeigt, dass die Plattform im Grundsatz gut navigierbar und benutzerfreundlich ist. Einzelne Funktionen und Konzepte (z.B. die Kursstruktur) benötigen jedoch noch eine klarere Kommunikation und visuelle Rückmeldung, um Missverständnisse zu vermeiden. Die gewonnenen Erkenntnisse können in weitere Entwicklungs-Iterationen der Plattform eingeflossen werden.

Ein weiterer iterativer Test mit mehreren Testpersonen wäre sinnvoll, um diese Hypothesen weiter zu validieren und quantitative Aussagen zur Usability treffen zu können.

5.10. Betriebskonzept

Dieses Unterkapitel zeigt auf, wie Securaware in der Produktionsumgebung betrieben wird. Das Konzept wird anhand von Abbildung 16 Form eines Deploymentdiagramms [50] visualisiert.

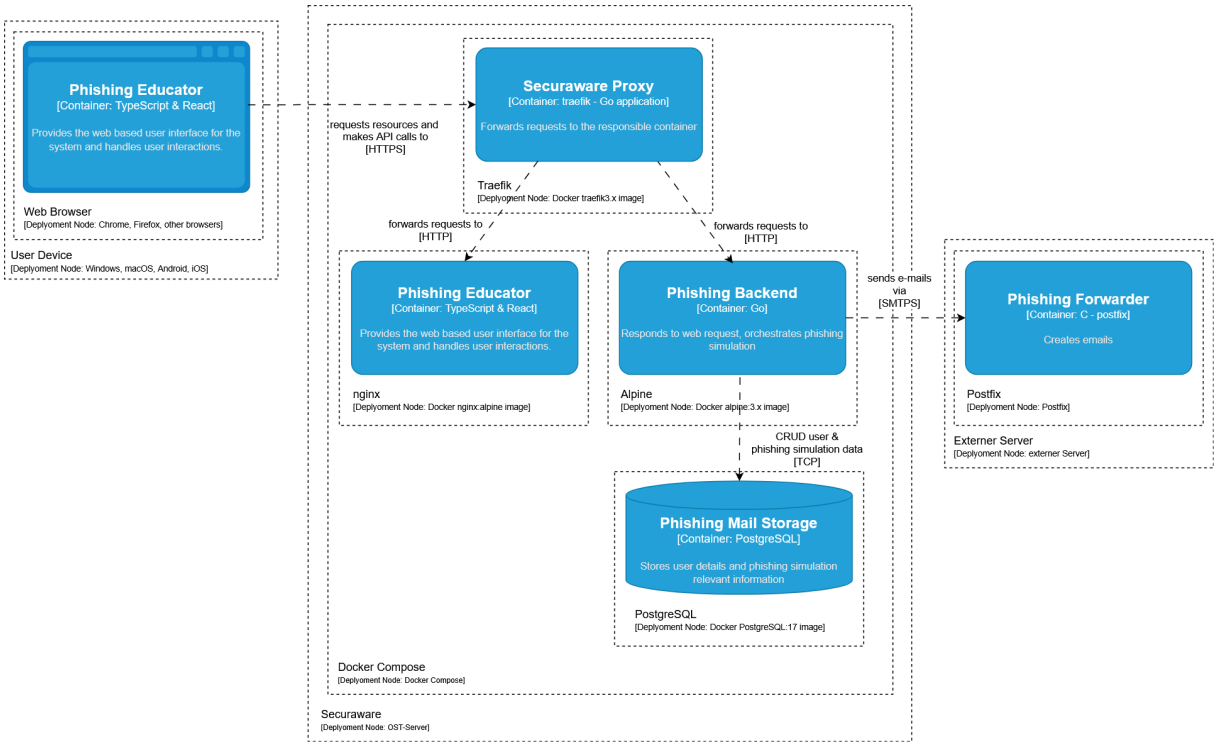


Abbildung 16: Betriebskonzept von Securaware in Form eines Deploymentdiagramms

Wie bereits beschrieben, sind die meisten C4-Container dockerisiert, sodass Securaware simple auf anderen Servern betrieben werden kann. Der Phishing Forwarder wird auf einem externen Server betrieben, sodass die E-Mails der Phishing-Simulation nicht auf die IP-Addresse der OST zurückgeführt werden, wodurch potenzielle Schäden, die beispielsweise durch eine Aufnahme der Server-IP in öffentliche Blacklists verursacht werden könnten, verhindert werden können.

5.11. Continuous Integration und Continuous Delivery (CI/CD)

Der Securaware wird in einem OST-internen GitLab-Repository verwaltet. Entsprechend werden für CI/CD GitLab-Jobs verwendet. Die entsprechende Pipeline für Main-Builds wird anhand Abbildung 17 visualisiert.

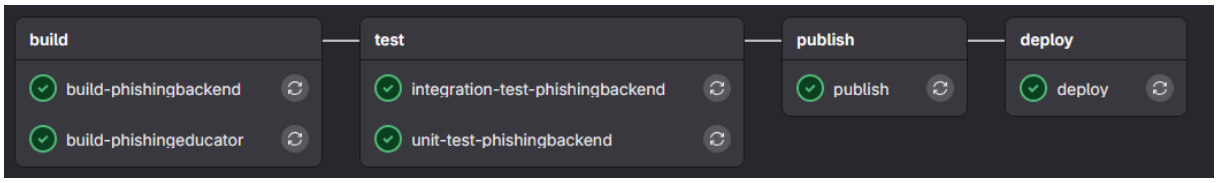


Abbildung 17: Continuous Integration und Continuous Delivery Main-Build-Pipeline

Ziel der Main-Pipeline ist es, zu überprüfen, ob der Phishing Educator und das Backend kompiliert werden können, alle automatisierten Testfälle erfolgreich ausgeführt werden können und die neuste Version von Securaware live zu schalten.

Diese Zielen werden in verschiedenen Stages verfolgt:

- Build: Verifiziert, dass der Phishing Educator und das Backend kompiliert werden können.

- Test: Verifiziert, dass alle automatisierten Testfälle erfolgreich ausgeführt werden können.
- Publish: Erstellt Docker-Images des Educators und Backends und lädt diese in die GitLab-Registry.
- Deploy: Verlinkt die zuvor erstellten Images in Docker-Konfigurationen, verbindet sich mit dem OST-Server, kopiert die Docker-Konfigurationen mitsamt den Verweisen zu der GitLab-Registry und startet das Docker-Compose-Cluster.

5.11.1. Test-Stage

Die Test-Stage verifiziert, dass alle automatisierten Testfälle erfolgreich ausgeführt werden können. Code 10 zeigt die Konfigurationen der Jobs auf.

```yaml
unit-test-phishingbackend:
  stage: test
  image: golang:1.24.1
  script:
    - cd PhishingBackend
    - go run gotest.tools/gotestsum@latest --junitfile testResults.xml --packages $(go list ./... | grep -vE '/integration_tests')
  artifacts:
    when: always
    paths:
      - PhishingBackend/testResults.xml
    reports:
      junit: PhishingBackend/testResults.xml

integration-test-phishingbackend:
  stage: test
  image: golang:1.24.1
  variables:
    POSTGRES_USER: testuser
    POSTGRES_PASSWORD: test
    POSTGRES_DB: testdb
  services:
    - postgres:latest
  script:
    - cd PhishingBackend/integration_tests
    - set -a
    - source setup/integrationTests.env
    - PHBA_DB_HOST=postgres
    - set +a
    - go run gotest.tools/gotestsum@latest --junitfile testResults.xml --packages $(go list -tags integration ./...) -- -tags integration
  artifacts:
    when: always
    paths:
      - PhishingBackend/integration_tests/testResults.xml
    reports:
      junit: PhishingBackend/integration_tests/testResults.xml
```

Code 10: GitLab-Jobs für automatisierte Testfälle

- Zeile 6 & 29: GitLab erwartet die Testresultate im JUnit-Report-Format. Die Testfälle von Securaware werden in Golang geschrieben und ausgeführt. Golang unterstützt dieses Format nicht. Die externe Softwarebibliothek gotest.tools/gotestsum wird verwendet, welche dieses Format hingegen unterstützt.
- Zeilen 7 - 12 & 30 - 35: Die Testresultate werden referenziert, sodass GitLab diese erfassen und visualisieren kann.
- Zeile 21 & 22: Die Integrationtests erwarten eine PostgreSQL-Instanz. Eine entsprechende Docker-Instanz wird wegen diesen Zeilen gestartet.

- Zeilen 24 - 28: Die Integrationtests erwarten Umgebungsvariablen. Diese werden über diese Konfigurationen aus integrationTests.env gelesen und gesetzt. Des Weiteren wird in Zeile 27 die Umgebungsvariable für die Datenbankurl angepasst, sodass diese auf die PostgreSQL-Docker-Instanz von Zeile 21 & 22 zeigt.

5.11.2. Publish und Deploy-Stage

Diese Stages sind für die Auslieferung der neusten Version der Securaware Software in die Produktionsumgebung zuständig. Code 11 zeigt die Konfigurationen der Jobs auf.

```
1  publish:
2    stage: publish
3    rules:
4      - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
5    services:
6      - name: docker:25.0.3-dind
7        alias: docker
8    before_script:
9      - echo "${CI_REGISTRY_PASSWORD}" | docker login -u
   ${CI_REGISTRY_USER} --password-stdin ${CI_REGISTRY}
10     - echo CI_REGISTRY_IMAGE=${CI_REGISTRY_IMAGE} >> ./.docker/.env
11     - echo CI_COMMIT_SHA=${CI_COMMIT_SHA} >> ./.docker/.env
12   script:
13     - docker compose -f .docker/compose.yaml build
14     - docker compose -f .docker/compose.yaml push
15
16 deploy:
17   stage: deploy
18   needs:
19     - job: publish
20   rules:
21     - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
22   before_script:
23     - echo CI_REGISTRY_IMAGE=${CI_REGISTRY_IMAGE} >> ./.docker/.env
24     - echo CI_COMMIT_SHA=${CI_COMMIT_SHA} >> ./.docker/.env
25   script:
26     - eval $(ssh-agent -s)
27     - echo "$CD_PRIVATE_KEY" | tr -d '\r' | ssh-add -
28     - mkdir -p ~/.ssh
29     - chmod 700 ~/.ssh
30     - scp -o StrictHostKeyChecking=no ./.docker/compose.yaml ./.docker/
   compose.prod.yaml ./.docker/.env ./.docker/create-extension.sql
   "$CD_USERNAME"@"$CD_HOST":/home/$CD_USERNAME/securaware/.docker
31     - ssh -o StrictHostKeyChecking=no "$CD_USERNAME"@"$CD_HOST" "docker
   compose -f /home/$CD_USERNAME/securaware/.docker/compose.yaml -f /home/
   $CD_USERNAME/securaware/.docker/compose.prod.yaml up -d"
```

Code 11: GitLab-Jobs für continuous delivery

- Zeile 9: Mit den OST-Gitlab-Benutzerdaten wird sich in Docker eingeloggt. Dies ist notwendig, um in einem späteren Schritt die Docker-Images später ins Registry hochzuladen
- Zeilen 10 & 11: Die Adresse der Registry und ein Identifier werden in eine .env-Datei gespeichert.
- Zeilen 13 & 14: Die Docker-Images werden erzeugt und in die Registry hochgeladen.
- Zeilen 26 - 28: SSH-Authentifizierungsdetails werden erstellt
- Zeile 30: Docker-Konfigurationen werden auf den Server hochgeladen.
- Zeile 31: Der Docker-Cluster wird gestartet. Der Cluster lädt über die Verweise aus Zeile 10 und 11 die Docker-Images herunter, erstellt Container und startet diese.

5.12. Betriebskonzept nach der Bachelorarbeit

Wie wird Securaware veröffentlicht? Wie werden neue Features freigegeben? Darf aus rechtlicher Sicht Securaware veröffentlicht werden? Diese und weitere Fragen werden in diesem Unterkapitel beantwortet.

Securaware ist vollständig virtualisiert mit Docker-Containern. Entsprechend entsteht keine Abhängigkeit zu einem „Infrastructure“ oder „Platform as a Service“-Anbieter. Ein „Vendor lock-in“ wird somit verhindert. Entsprechend können Angebote verschiedener Anbieter verglichen werden. Securaware kann somit ohne grossen Aufwand ausserhalb der OST veröffentlicht werden.

Securaware soll über eine ansprechende Domain aufgerufen werden können. Die Domain „www.securaware.ch“ ist bereits im Besitz des Teams und wird bereits verwendet. Securaware kann entsprechend bereits vermarktet werden.

Durch Docker können neue Features über Nacht über eine „Big Bang Migration“ aufgeschalten werden. Durch den Datenbankmigrator können Änderungen am Datenbankschema ohne Datenverlust ausgeführt werden.

Für die kommerzielle Nutzung von Securaware sind unter anderem die Lizenzen der verwendeten Software entscheidend. Diese Lizenzen können diese kommerzielle Nutzung einschränken. Die Lizenzen sind im Anhang aufgelistet.

6. Diskussion und Schlussfolgerung

Dieses Kapitel beantwortet mithilfe einer Zusammenfassung der Resultate, sowie einem Vergleich mit den Anforderungen, die in Abschnitt 2.3 definierte Forschungsfrage. Ausserdem werden Limitationen aufgezeigt und ein Ausblick für zukünftige Arbeiten gegeben.

6.1. Resultate

Die Resultate dieser Arbeit werden in diesem Unterkapitel zusammengefasst.

- **Phishing Simulation:** Personalisierte und kategorisierte Phishing-E-Mails, welche sich dem Kompetenzniveau des Benutzers anpassen, wurden erstellt. Diese simulieren Phishing-Angriffe praxisnah und bereiten Benutzer optimal auf wirkliche Angriffe vor.
- **Online-Kurse:** Lernende können ihr eigenes Tempo bestimmen, Inhalte wiederholen und schwierige Themen intensiver bearbeiten in interaktiven Online-Kursen.
- **Tests:** Das durch die Online-Kurse erlange Wissen kann über diese Tests überprüft und vertieft werden.
- **Gamifizierung:** Die Gamifizierung sorgt dafür, dass Benutzer weiterhin motiviert sind, sich mit Phishing auseinanderzusetzen und Neues zu lernen.
- **Reminder:** Neben der Gamifizierung werden Erinnerungs-E-Mails versendet, welche den Benutzern die Gefahren von Phishing erläutert.
- **Dokument:** Dieses Dokument beschreibt die Ausgangslage, das Ziel, die Methodik, die Umsetzung und selbstverständlich das Produkt. Lösungen, Hindernisse, neu Gelerntes, Praktiken, und weiteres werden dem Leser vermittelt.
- **Erfahrung und Wissen:** Durch diese Arbeit durften die Projektteilnehmer in Themen wie unter anderem Phishing, Golang und Software-Engineering im Allgemeinen vertiefen.

Zusammengefasst bietet der entwickelte Prototyp ein solides Fundament einer, wie in der Forschungsfrage definierten, Schulungsplattform. Eine Evaluierung der Effektivität des Produkts ist nicht Teil dieser Arbeit und würde die Rahmenbedingungen (siehe Abschnitt 1.3) der Arbeit übersteigen. Stattdessen wurde der Fokus auf die Konzeption und die technische Umsetzung gesetzt.

6.2. Diskussion

In diesem Unterkapitel werden die implementierten Features mit den Anforderungen verglichen.

Resultat	Vergleich mit Anforderungen
Phishing Simulation	Die Phishing Simulation ist persönlich, passt sich dem Wissensstand des Benutzers an und notiert sich die Schwachstellen des Benutzers. Falsch eingeschätzte Simulationen werden dem Benutzer erläutert mitsamt Hinweisen, welche auf Spam schliessen. Das Feature erfüllt die Anforderungen.
Online-Kurse	Die Kurse sensibilisieren den Benutzer interaktiv für Phishing. Sie sind ansprechend gestaltet und halten den Fortschritt des Benutzers fest. Es wurden nicht alle Kurse implementiert. Zusätzlich fehlt ein unique Sellingpoint, mit welchem sich die Kurse von anderen Lernplattformen abheben.
Tests	Die Tests prüfen das Wissen der Benutzer und zeigen nach Abschluss sowohl die korrekten als auch die falsch beantworteten Fragen an. Durch die einfache Erweiterbarkeit mittels zusätzlicher Datenbankeinträge lassen sie sich gut skalieren. Ein Nachteil besteht jedoch darin, dass die Tests ausschliesslich textbasiert sind und keine visuellen Elemente enthalten.
Gamifizierung	Die Gamifizierung bildet eine solide Grundlage und kann durch zusätzliche Elemente wie freischaltbare Profilbilder oder eine Sharing-Funktion für soziale Medien sinnvoll erweitert werden.
Reminder	Der Reminder erfüllt seinen Zweck, könnte jedoch persönlicher sein und sich auf die Aktionen des Benutzers beziehen. Ein Beispiel hierfür wäre den letzten Kurs, Test oder Phishing Simulation im E-Mail zu erwähnen.

Tabelle 27: Dikussion, Vergleich der Anforderungen und Resultate

6.3. Limitationen & Ausblick

In diesem Unterkapitel werden Funktionalitäten, Anregungen und Ideen, welche nicht umgesetzt werden konnten, beleuchtet.

- **Phishing-Mentor**: Der in Abschnitt 4.2.7 beschriebene Phishing-Mentor konnte nicht umgesetzt werden. Hauptgrund hierfür ist die späte Ideenfindung, wodurch das Feature erst spät im Projektverlauf ausgearbeitet wurde. Wie im angesprochenen Kapitel beschrieben, ist dieses Feature gemäss den Recherchen einzigartig und ist nicht in anderen Produkten enthalten. Entsprechend hat dies eine hohe Bedeutung und soll vor der Veröffentlichung von Securaware implementiert werden.
- **Rangliste**: Aus Zeitgründen konnte die Rangliste während der Umsetzungs-Phase nicht mehr umgesetzt werden. Ein Gamification-Mechanismus besteht bereits, jedoch kann ohne eine Rangliste oder ein vergleichbares Feature nicht der vollständige Nutzen der Funktion herausgeholt werden. Es ist daher unerlässlich, dies noch vor einer Veröffentlichung von Securaware zu implementieren.
- **Refresher**: Der gemäss Abschnitt 4.2.2 beschriebene Refresher wurde nicht implementiert aufgrund des Aufwandes.

- **WhatsApp-Community**: Die in Abschnitt 4.2.6 WhatsApp-Community kann erstellt werden, sobald Securaware kommerziell verwendet wird. Diese Community wirbt neue Benutzer an und motiviert existierende Benutzer wieder Securaware zu verwenden.

7. Bibliographie

[1] kimmin1kk, „[REST API] HATEOAS“. Zugegriffen: 5. Mai 2025. [Online]. Verfügbar unter: https://velog.io/@kimmin1kk/REST-API-HATEOAS

[2] Jeffrey Palermo, „Onion Architecture“. Zugegriffen: 21. April 2025. [Online]. Verfügbar unter: https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/

[3] Robert C. Martin, „Clean Architecture“. Zugegriffen: 21. April 2025. [Online]. Verfügbar unter: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

[4] Wikipedia, „Hexagonal Architecture“. Zugegriffen: 21. April 2025. [Online]. Verfügbar unter: https://en.wikipedia.org/wiki/Hexagonal_architecture_(software)

[5] Jimmy Bogard, „Vertical Slice Architecture“. Zugegriffen: 21. April 2025. [Online]. Verfügbar unter: https://www.jimmybogard.com/vertical-slice-architecture/

[6] Traefik Labs, „traefik/traefik: The Cloud Native Application Proxy“. Zugegriffen: 31. Mai 2025. [Online]. Verfügbar unter: https://github.com/traefik/traefik

[7] Xun Dong, John A. Clark, und Jeremy Jacob, „Modelling User-Phishing Interaction“, *2008 Conference on Human System Interactions*, S. 627–632, 2008, doi: 10.1109/HSI.2008.4581513.

[8] Ahmed Aleroud und Lina Zhou, „Phishing environments, techniques, and countermeasures: A survey“, *Computers & Security 68*, S. 160–196, 2017, doi: 10.1016/j.cose.2017.04.006.

[9] Maksim Sharabov, Georgi Tsochev, Veska Gancheva, und Antoniya Tasheva, „Filtering and Detection of Real-Time Spam Mail Based on a Bayesian Approach in University Networks“, Bd. 374(2), S. 94–123, Jan. 2016, doi: 10.3390/electronics13020374.

[10] Jordan Crain, Lukasz Opyrchal, und Atul Prakash, „Fighting Phishing with Trusted Email“, *2010 International Conference on Availability, Reliability and Security*, S. 462–467, 2010, doi: 10.1109/ARES.2010.98.

[11] Ivar Jacobson, Ian Spence, und Brian Kerr, „Use-Case 2.0: The Hub of Software Development“, *acmqueue*, Bd. 14(1), S. 94–123, Jan. 2016, doi: 10.1145/2898442.2912151.

[12] Simon Brown, *The C4 model for visualising software architecture*. Leanpub, 2019.

[13] Prathamesh Muzumdar, Amol Bhosale, Ganga Prasad Basyal, und George Kurian, „Navigating the Docker Ecosystem: A Comprehensive Taxonomy and Survey“, *Asian Journal of Research in Computer Science*, Bd. 17, S. 42–61, 2024, doi: 10.9734/ajrcos/2024/v17i1411.

[14] W3C, „Web Content Accessibility Guidelines (WCAG) 2.1“. Zugegriffen: 26. März 2025. [Online]. Verfügbar unter: https://www.w3.org/TR/WCAG21/

[15] Prof. Dr. Olaf Zimmermann, „ZIOL-AA2024-W3SolutionStrategyCSCsv10p“. 2024.

[16] Wikipedia, „Server Name Indication“. Zugegriffen: 18. April 2025. [Online]. Verfügbar unter: https://en.wikipedia.org/wiki/Server_Name_Indication

[17] Mark Nottingham, Erik Wilde, und Sanjay Dalal, „RFC 9457 - Problem Details for HTTP APIs“, Juli 2023, *Internet Engineering Task Force (IETF)*. doi: https://doi.org/10.17487/RFC9457.

[18] OpenJS Foundation und jQuery contributors, „jQuery“. Zugegriffen: 4. April 2025. [Online]. Verfügbar unter: https://jquery.com/

[19] Astro contributors, „Astro“. Zugegriffen: 4. April 2025. [Online]. Verfügbar unter: https://astro.build/

[20] Google, „AngularJS — Superheroic JavaScript MVW Framework“. Zugegriffen: 4. April 2025. [Online]. Verfügbar unter: https://angularjs.org/

[21] Meta Platforms Inc, „React“. Zugegriffen: 4. April 2025. [Online]. Verfügbar unter: https://react.dev/

[22] Evan You, „Vue.js - The Progressive JavaScript Framework“. Zugegriffen: 4. April 2025. [Online]. Verfügbar unter: https://vuejs.org/

[23] Vercel und Svelte contributors, „Svelte • Web development for the rest of us“. Zugegriffen: 4. April 2025. [Online]. Verfügbar unter: https://svelte.dev/

[24] Team CodeReliant, „Battle of the Frameworks: Benchmarking High-Performance HTTP Libraries“. Zugegriffen: 15. April 2025. [Online]. Verfügbar unter: https://www.codereliant.io/p/battle-of-the-frameworks-benchmarking-high-performance-http-libraries

[25] True Facts, „2024's Fastest Web Servers for REST APIs: Node.js vs Go vs Rust vs C# (.NET) Benchmark“. Zugegriffen: 15. April 2025. [Online]. Verfügbar unter: https://medium.com/%40hiadeveloper/2024s-fastest-web-servers-for-rest-apis-node-js-vs-go-vs-rust-vs-c-net-benchmark-665d8efd2f44

[26] Oracle, „Java Software | Oracle“. Zugegriffen: 4. April 2025. [Online]. Verfügbar unter: https://www.oracle.com/java/

[27] Microsoft, „C# | Modern, open-source programming language for .NET“. Zugegriffen: 4. April 2025. [Online]. Verfügbar unter: https://dotnet.microsoft.com/en-us/languages/csharp

[28] Google, „The Go Programming Language“. Zugegriffen: 4. April 2025. [Online]. Verfügbar unter: https://go.dev/

[29] „Rust Programming Language“. Zugegriffen: 4. April 2025. [Online]. Verfügbar unter: https://www.rust-lang.org/

[30] OpenJS Foundation, „Node.js - Run JavaScript Everywhere“. Zugegriffen: 4. April 2025. [Online]. Verfügbar unter: https://nodejs.org/en

[31] Linux Foundation, „CNCF Landscape“. Zugegriffen: 18. April 2025. [Online]. Verfügbar unter: https://landscape.cncf.io/?group=projects-and-products

[32] Linux Foundation, „OpenAPI“. Zugegriffen: 19. April 2025. [Online]. Verfügbar unter: https://www.openapis.org/

[33] R. T. Fielding, „Architectural Styles and the Design of Network-based Software Architectures“, Doctoral dissertation, University of California, Irvine, 2000.

[34] Leonard Richardson und Sam Ruby, *RESTful Web Services*, 1. Aufl. O'Reilly Media, 2007.

[35] Robert C. Martin und Kevlin Henney, *Clean Architecture: A Craftsman's Guide to Software Structure and Design (Robert C. Martin Series)*, 1. Aufl. Pearson, 2017.

[36] Alistair Cockburn, „Hexagonal Architecture“. Zugegriffen: 21. April 2025. [Online]. Verfügbar unter: https://alistair.cockburn.us/hexagonal-architecture

[37] Martin Fowler, *Patterns of Enterprise Application Architecture*, 1. Aufl. Addison-Wesley Professional, 2002.

[38] Google, „Frequently Asked Questions (FAQ) - The Go Programming Language“. Zugegriffen: 29. April 2025. [Online]. Verfügbar unter: https://go.dev/doc/faq#How_do_I_write_a_unit_test

[39] Jamie Tanna und oapi-codegen, „oapi-codegen/oapi-codegen: Generate Go client and server boilerplate from OpenAPI 3 specifications“. Zugegriffen: 20. Mai 2025. [Online]. Verfügbar unter: https://github.com/oapi-codegen/oapi-codegen/

[40] „The Postfix Home Page“. Zugegriffen: 4. April 2025. [Online]. Verfügbar unter: https://www.postfix.org/start.html

[41] Electronic Frontier Foundation, „Certbot“. Zugegriffen: 13. April 2025. [Online]. Verfügbar unter: https://certbot.eff.org/

[42] Internet Security Research Group, „Let's Encrypt“. Zugegriffen: 13. April 2025. [Online]. Verfügbar unter: https://letsencrypt.org/

[43] Traefik Labs, „Traefik Proxy“. Zugegriffen: 31. Mai 2025. [Online]. Verfügbar unter: https://doc.traefik.io/traefik/

[44] Docker Inc, „Networking with standalone containers | Docker Docs“. Zugegriffen: 11. April 2025. [Online]. Verfügbar unter: https://docs.docker.com/engine/network/tutorials/standalone

[45] Docker Inc, „Multi-stage | Docker Docs“. Zugegriffen: 12. April 2025. [Online]. Verfügbar unter: https://docs.docker.com/build/building/multi-stage/

[46] Google, „go command - cmd/go - Go Packages“. Zugegriffen: 13. April 2025. [Online]. Verfügbar unter: https://pkg.go.dev/cmd/go#hdr-Build_and_test_caching

[47] Docker Inc, „Optimize cache usage in builds | Docker Docs“. Zugegriffen: 13. April 2025. [Online]. Verfügbar unter: https://docs.docker.com/build/cache/optimize/#use-cache-mounts

[48] Melanie Diepenbeck, Ulrich Kühne, Mathias Soeken, und Rolf Drechsler, „Behaviour Driven Development for Tests and Verification“, *Tests and Proofs*, Bd. 8, S. 61–77, 2014, doi: 10.1007/978-3-319-09099-3_5.

[49] Jakob Nielsen, „Thinking Aloud: The #1 Usability Tool“. Zugegriffen: 12. Juni 2025. [Online]. Verfügbar unter: https://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/

[50] Simon Brown, „Deployment diagram“. Zugegriffen: 1. Juni 2025. [Online]. Verfügbar unter: https://c4model.com/diagrams/deployment