WEITERENTWICKLUNG UND WEBSECURITY DES LABALIVE IDENTITY- UND ACCESMANAGEMENTS

BACHELORARBEIT

IM RAHMEN DES STUDIENGANGES

TECHNISCHE INFORMATIK UND KOMMUNIKATIONSTECHNIK

Niklas Dombek

Betreuer:

Prof. Dr.-Ing. Erwin Riederer

Tag der Abgabe: 30.06.2023

Universität der Bundeswehr München
Fakultät für Elektrotechnik und Technische Informatik
Institut für Funkkommunikation

Neubiberg, Juni 2023

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, insbesondere keine anderen als die angegebenen Informationen.

Der Speicherung meiner Bachelorarbeit zum Zweck der Plagiatsprüfung stimme ich zu. Ich versichere, dass die elektronische Version mit der gedruckten Version inhaltlich übereinstimmt.

Neubiberg, den 30.06.2023

_____

Niklas Dombek

Abstract

Die Arbeit befasst sich mit der Implementierung eines neuen, sichereren und schnelleren Verfahrens zur Speicherung und Verarbeitung von benutzerspezifischen Daten, die sich auf die Inhalte der Webseite labAlive beziehen. Zudem galt es das Access Management zu überarbeiten und ein neues Login Verfahren zu implementieren, bei dem eine klassische Nutzername und Passwort Kombination umgesetzt werden sollte. Ebenfalls wird ein Ausblick auf weitere Umsetzungen geben, die dem Identity- und Access Management sowie der Websecurity zu Gute kommen würden.

Inhaltsverzeichnis

1 Einleitung

In der heutigen digitalen Ära, in der Informationen und Dienste zunehmend über das Internet zugänglich sind, wird die Sicherheit von Daten zu einer immer größeren Herausforderung. Unternehmen und Organisationen setzen vermehrt auf Identity- und Accessmanagement (IAM)-Systeme, um den Zugriff auf ihre Ressourcen zu verwalten und sicherzustellen, dass nur autorisierte Benutzer darauf zugreifen können.

Diese Bachelorarbeit befasst sich mit der Weiterentwicklung und Websecurity des labAlive Identity- und Access Managementsystems. Dabei liegt der Fokus auf der Weiterentwicklung des bereits vorhandenen Grundbausteines und dem Ausbau des Ressourcenmanagements für Benutzer. Im Rahmen dieser Arbeit, wird das bereits bestehende Login-Verfahren überarbeitet und an die heutigen Standards, sowohl aus Komfortsicht, als auch aus sicherheitlicher Perspektive, angepasst.

Die Webseite labAlive [1] bietet Usern die Möglichkeit, rechnergestützte Simulationen online auszuführen. Dies bietet nicht nur für Studenten der technischen Informatik und Kommunikationstechnik im Studium einen enormen Mehrwert, sondern auch für jeden Kommunikationstechniker, dem das Labor für eine Signalanalyse, oder die Ressourcen für eine gute Simulation, fehlen.

Diese Arbeit trägt dazu bei, das Ressourcenmanagements sowie das Identity- und Accessmanagement der Webseite labAlive weiterzuentwickeln und die Websecurity der Plattform zu stärken. Durch den gesetzten Fokus und die daraus entstehende Implementierung können Benutzer von labAlive ihre Projekte effizienter und sicherer gestalten, wodurch der Mehrwert und die Zuverlässigkeit der Plattform weiter gesteigert werden.

2 Verwendete Software und Programmiersprachen

2.1 Software

2.1.1 IntelliJ IDEA

IntelliJ IDEA ist eine leistungsstarke Java-Entwicklungsumgebung (IDE) mit benutzerfreundlicher Oberfläche und umfangreichen Funktionen für effiziente Programmierung, einschließlich Code-Vervollständigung, Refactoring und Fehlererkennung. [2]

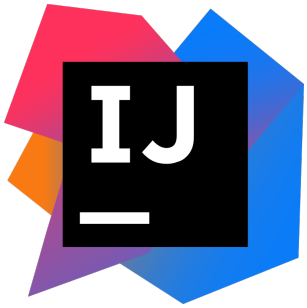

Abbildung 2.1: IntelliJ-Logo[3]

2.1.2 Gitea

Gitea ist eine Open-Source-Plattform zur Verwaltung von Git-Repositories. Es bietet eine benutzerfreundliche Weboberfläche und Funktionen für die Versionsverwaltung, Zusammenarbeit und Issue-Tracking. Gitea ist eine leichtgewichtige Alternative zu anderen Git-Hosting-Plattformen wie GitHub oder GitLab. [4]

Abbildung 2.2: Gitea-Logo[5]

2.1.3 Visual Studio Code

Visual Studio Code ist ein plattformübergreifender, kostenloser Code-Editor von Microsoft. Mit IntelliSense, integriertem Git und Debugging-Tools ist er ideal für die Entwicklung von Webanwendungen und bietet eine Vielzahl von Erweiterungen für zusätzliche Funktionalität. [6]

Abbildung 2.3: Visual Studio Code Logo[7]

2.1.4 Microsoft Visio

Microsoft Visio ist eine Diagramm- und Visualisierungssoftware zur Erstellung von klaren und ansprechenden Diagrammen, Flussdiagrammen und Organigrammen. Es bietet eine benutzerfreundliche Oberfläche und eine breite Palette von Vorlagen und Tools, um komplexe Informationen visuell darzustellen und zu kommunizieren. [8]

Abbildung 2.4: Microsoft VisioLogo[9]

2.1.5 Microsoft Access

Microsoft Access ist eine benutzerfreundliche Datenbanksoftware von Microsoft, die es Benutzern ermöglicht, Datenbanken zu erstellen, zu verwalten und darauf zuzugreifen. Es bietet Funktionen wie Tabellen, Abfragen, Formulare und Berichte zur Organisation und Analyse von Daten. [10]

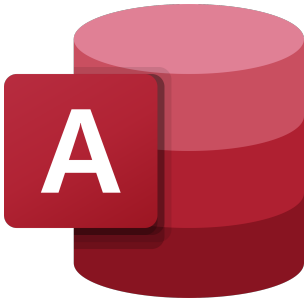

Abbildung 2.5: Microsoft AccessLogo[11]

2.2 Programmiersprachen

2.2.1 Java

Java ist eine plattformunabhängige, objektorientierte Programmiersprache mit einer robusten Laufzeitumgebung. Sie ist für ihre Einfachheit, Lesbarkeit und breite Anwendungspalette bekannt und wird in verschiedenen Bereichen wie Desktop-, Web- und mobilen Anwendungen eingesetzt. [12]

Abbildung 2.6: Java-Logo[13]

2.2.2 HTML

HTML (Hypertext Markup Language) ist eine grundlegende Markup-Sprache zur Erstellung von Webseiten. Es verwendet Tags, um den Inhalt zu strukturieren und zu formatieren, und wird von Webbrowsern interpretiert, um die Webseite anzuzeigen. [14]

2.2.3 JavaScript

JavaScript ist eine interpretierte Programmiersprache, die für die Entwicklung von interaktiven Webseiten verwendet wird. Sie ermöglicht das Hinzufügen von dynamischen Inhalten und wird sowohl auf der Client- als auch auf der Serverseite eingesetzt. [15]

2.2.4 SQL

SQL (Structured Query Language) ist eine spezielle Sprache zur Verwaltung und Abfrage von Datenbanken. Sie ermöglicht das Erstellen, Ändern und Abfragen von Datenbanktabellen, das Einfügen, Aktualisieren und Löschen von Daten sowie komplexe Operationen wie Joins und Aggregatfunktionen. SQL ist ein wesentlicher Bestandteil der Datenbankentwicklung und -verwaltung und wird von den meisten relationalen Datenbanksystemen unterstützt. [16]

3 Gegenstand der Arbeit

3.1 Aufgabenstellung

Das konkrete Ziel ist es ein System zu implementieren, welches die Wartezeit bei Zugriffen auf eigene Daten verkürzt, diese zugleich sicher speichert und nur autorisierten Usern Zugriff gewährt. Dazu zählt ebenfalls die Erweiterung der Webseite, um die Bearbeitung- und Erstellungsmöglichkeit aller vorhandenen Ressourcen, die labAlve zu bieten hat, um den vollen Funktionsumfang der Webseite individuell nutzbar zu machen.

Ebenfalls soll ein sicherer Login Prozess umgesetzt werden, bei dem klassisch ein Nutzername und ein Passwort verwendet wird. Die Logindaten müssen dabei sicher in der vorhandenen Datenbank gespeichert werden. Aufgabe ist, lediglich den Login Prozess anzupassen und die nachfolgende Verarbeitung mittels des erstellten Tokens nicht zu verändern, sondern nur zu erweitern. Daher muss auch das Mapping der Nutzername-Passwort-Kombination auf das erstellte Token sicher implementiert werden, um das Token weiterhin zur Nutzeridentifikation mittels Cookies zu gewährleisten.

3.2 Überblick

Abbildung 3.1: labAlive Webseite

LabAlive ermöglicht die Simulation und grafische Darstellung verschiedener Übertragungsverfahren, Nachrichtensysteme sowie Signalmodulationen. Ähnlich einem realen Laborexperiment mit Messgeräten können diese Simulationen interaktiv genutzt werden, um Parameter anzupassen und die daraus resultierenden Auswirkungen visuell zu veranschaulichen.

Der aktuelle Stand der online Laborumgebung labAlive sieht im Bereich Identity- und Accessmanagement sowie der Ressourcenverarbeitung folgendermaßen aus. Ein Nutzer kann vorgefertigte Versuche ausführen und analysieren, aber nicht individuell anpassen. Zudem wurde bereits intern die Funktionalität umgesetzt, dass ein User spezielle Experimente erstellen und ausführen kann. Wenn dieser sich einen Account erstellt, kann er auch ausgewählte Ressourcen speichern und jederzeit wieder abrufen.

Nach einem erfolgreichen Registrierungsprozess, kann der Nutzer von den vorhanden Ressourcen, siehe Abb.[3.3], lediglich *Wirings* erstellen, bearbeiten und simulieren. Ebenfalls noch nicht implementiert, ist das Verlinken von Ressourcen, das Hinzufügen von Bildern zu einem Experiment sowie das Anlegen von Keywords zur schnelleren Suche.

Beim Registrieren erhält ein Nutzer einen automatisch generiertes Token, das die Webseite zur Identifikation mittels Cookies verwendet. Zudem ist dieses Token der einzige Weg sich auf der

Webseite anzumelden, wenn man sich ausloggt oder keinem persistenten Cookie zugestimmt hat. Ein Benutzername ist bisher optional. Siehe[3.2].

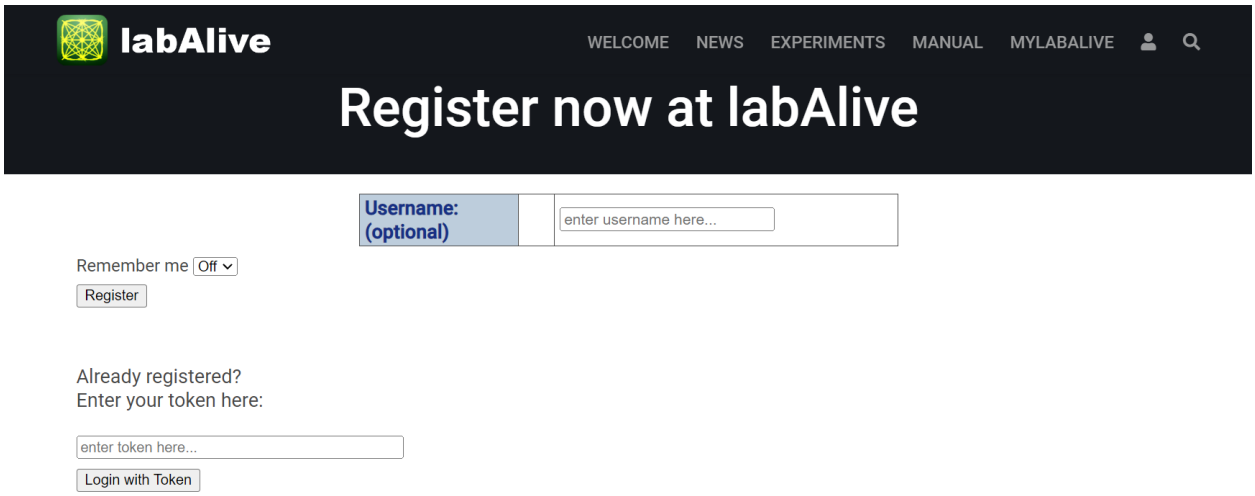

Abbildung 3.2: Alter Login labAlive

Die in Abb.[3.3] gezeigten Ressourcen bestehen bereits. Die abgebildeten Elemente, abgesehen von *Wirings*, bestehen jedoch nur im Code. Daher können sie aktuell nur von Entwicklern angepasst werden und nicht über eine grafische Oberfläche vom User erstellt und bearbeitet werden. Es gibt noch keine Datenbanktabellen zum Speichern von spezifischen Ressourcen.

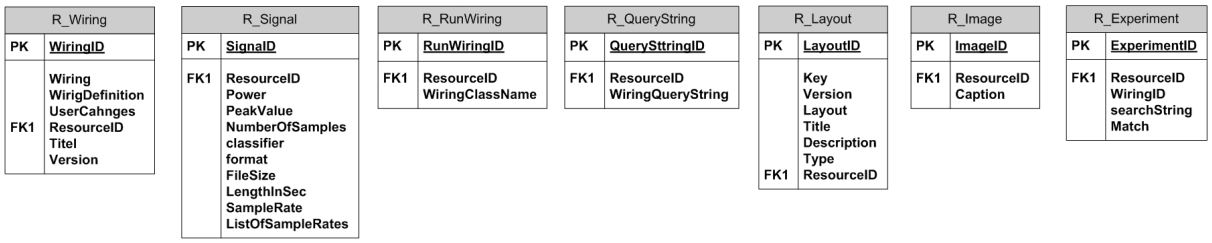

Abbildung 3.3: Ressourcen

Sowohl beim Registrierungsprozess als auch beim Anlegen, Bearbeiten und Löschen eines *Wirings* werden Web-Servlets benutzt, um Anfragen zu verarbeiten. Servlets sind Java-Klassen, die auf einem Webserver ausgeführt werden, um dynamische Inhalte für Webanwendungen zu generieren. Sie ermöglichen die Verarbeitung von Anfragen und die Erzeugung von entsprechenden Antworten auf der Serverseite. Servlets sind Teil der Java Enterprise Edition und dienen der Entwicklung und Erweiterung von plattformunabhängigen Webanwendungen. Mit diesen Servlets werden die Verarbeitung sowie Speicherung mittels weiterer Methoden und Klassen, welche im Folgenden ausführlich erklärt werden, bereitgestellt.

4 Ressourcen Verarbeitung

Die bereits vorhandene Ressourcenverarbeitung für *Wirings*, sollte nun auf alle Ressourcen ausgeweitet und effizienter gestaltet werden. Durch diese Umsetzungen wird ein erweiterter Grundbaustein bereitgestellt, welcher späteren Implementierungen des Identity- und Accessmanagements zugutekommt. Zusätzlich wird dadurch der Funktionsumfang als auch die Individualität der Webseite gesteigert.

4.1 Struktur der zu verwaltenden Ressourcen

Da Anfangs nur eine Struktur für die Speicherung der Daten eines Versuchstypen bestanden, musste dies auf effiziente und logische Weise für alle umgesetzt werden. Dabei war die Idee, dass alle darstellbaren und ausführbaren Experimente als Ressourcen angesehen werden, die am Ende ein User erstellen, bearbeiten und mit anderen Ressourcen verknüpft können soll.

Das bedeutet, dass alle Ressourcentypen, die in Abb. [4.1] auf der rechten Seite im Kasten zu sehen sind, eine Elternklasse namens *Resource* besitzen. Diese wird im folgenden auch Ressourcenkopf genannt. Die gezeigte Struktur ist zugleich der Aufbau der Datenbank, wodurch eine einheitliche Hierarchie entsteht.

In dieser Elternklasse werden die Metainformationen gespeichert. Siehe Abb. [4.1] Elternklasse *Resource*. Darunter fallen Parameter wie das letzte Speicherdatum, eine allgemeine Beschreibung der Ressource oder der Ressourcentyp, um bei vielen Funktionen zwischen den Ressourcen zu unterscheiden.

Die UserID ist ebenfalls in der Elternklasse enthalten, um Ressourcen eindeutig einem Nutzer, einer Nutzerin zuweisen zu können und später auch wirklich nur diese dem Anwender zu zeigen. Eine eindeutige Unterscheidung zwischen Usern wird durch ein eindeutiges Token gewährleistet. Dieser wird in einer separaten Klasse gespeichert, um eine Trennung zu Ressourcen-Daten zu ermöglichen. Auf diesen Punkt wird in Kapitel 5 genauer eingegangen.

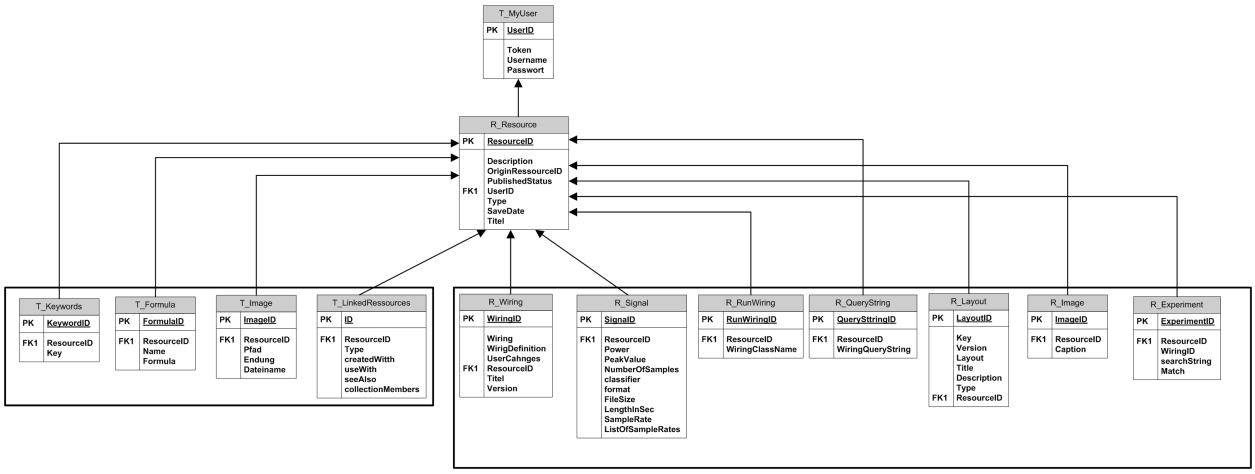

Abbildung 4.1: Resourcen Hirarchie

Neben den Ressourcentypen, die in Abb. [4.1], mit einem R_ gekennzeichnet sind, gibt es noch Ausprägungen, die Ressourcen erweitern sollen. Darunter sind unter anderem Keywords, die zur Filterung beziehungsweise Suche eingesetzt werden können, enthalten. Eine Auswahl aus einem Keyword Pool soll bei der Erstellung möglich sein.

Außerdem kann man Formeln und Bilder einbinden. Formeln sind spezifische Erweiterungen für die Ressource. Bilder sind Grafiken, die eingebunden werden können, um bei der Darstellung einen besseren Eindruck von der Art des Versuchs zu bekommen.

Die letzte Ausprägung, die man bei einer Ressource setzten kann, sind die *linked_Resources*. Dabei kann man in verschiedenen Arten Ressourcen miteinander verknüpfen. *Created With*, *Used With*, *See Also* und *Collection Members* sind dabei mögliche Verbindungen. *Created With* wird dazu verwendet, aufzuzeigen, dass eine Ressource mit einer oder mehreren anderen Ressourcen erstellt wurde. *Used With* wird genutzt, um in der Darstellung zu sehen, welche Ressourcen miteinander genutzt werden, um eine nachvollziehbare Struktur zu erhalten. *See Also* ist ein Verweis auf ähnliche Simulationen. Mit *Collection Members* soll es möglich sein, seine Ressourcen Collections zuzuweisen.

4.1.1 Frontend

Um die oben beschriebenen Ressourcen erstellen, bearbeiten und anpassen zu können, bedarf es eines Frontends. Dabei war es wichtig sicherzustellen, dass bestehende Funktionen für *Wirings* weiterhin bestehen bleiben. Es sollte möglich sein, sowohl Ressourcentypen und deren Daten, als auch den Ressourcenkopf und Ausprägungen individuell anzupassen.

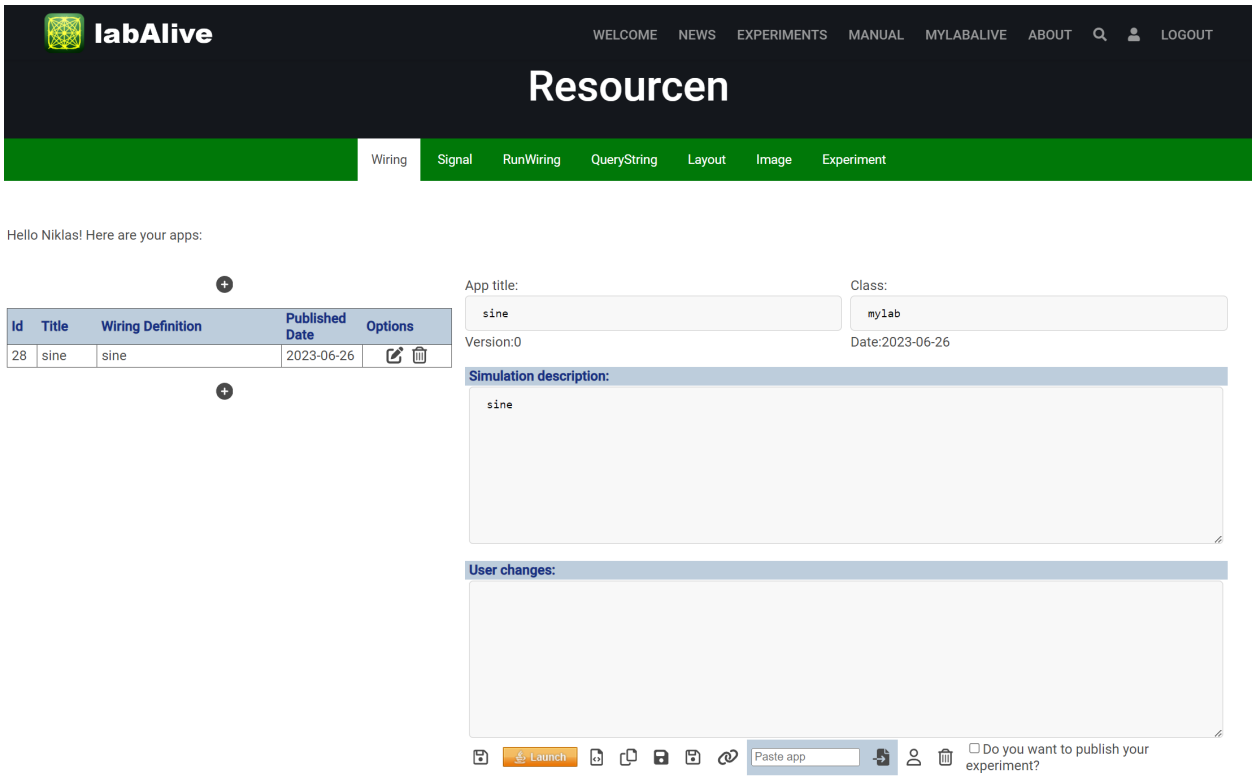

Abbildung 4.2: Resourcen Übersicht

In Abb. [4.2] sieht man wie die Übersicht der einzelnen Typen aussieht. Dabei kann man zu jedem Typ wechseln, eine neue Ressource erstellen und individuell anpassen.

Neben den Typen soll man in der Lage sein, den Ressourcenkopf und die Ausprägungen anzupassen. Dafür gibt es einen Button unter den Parametern, mit dem man auf eine weitere Seite wechseln kann, auf der man *Verlinkungen*, *Images*, *Keywords* und eine Beschreibung festlegen kann. [4.3] Bei den *Linked_Resources* trägt man die ResourceID ein und drückt *update Resource*. Damit wird automatisch eine Verknüpfung des gewünschten Typen mit einer anderen Ressource ermöglicht.

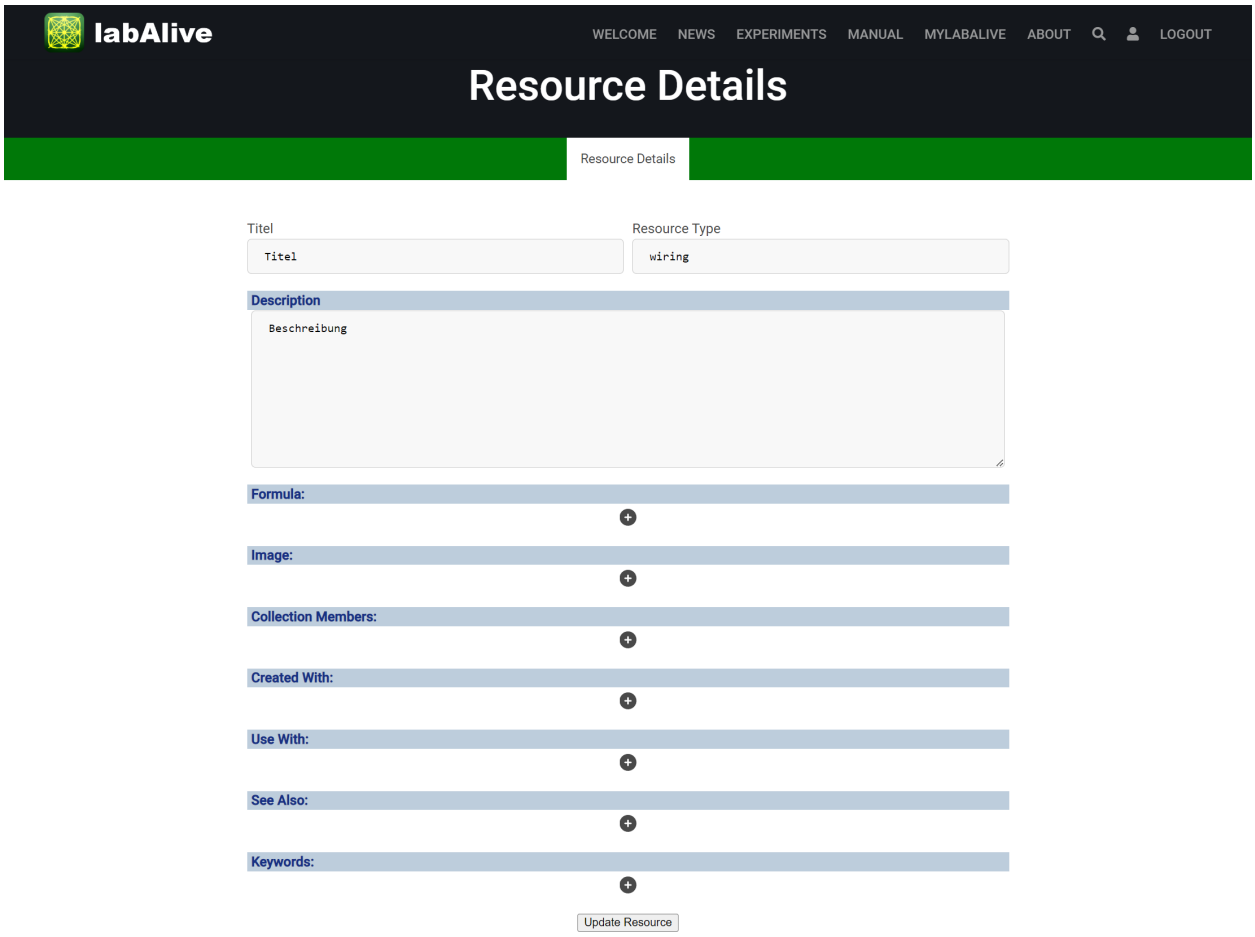

Abbildung 4.3: Metadaten und Verlinkungen

4.2 Planung und Konzeption der Speicherung

Um das Beschriebene umzusetzen, wird eine zwei-Säulen-Speicherung verwendet. Das bedeutet, dass die Daten persistent und laufzeitunabhängig in einer relationalen Datenbank in Access gespeichert werden. Die zweite Säule ist eine Art Cache, der zur Laufzeit, aus den Daten der Datenbank erstellt wird und so parallel zur Datenbank aktuell gehalten wird. Nach einem Serverneustart wird er erneut initialisiert.

Hintergrund dieses Verfahrens ist die leichtere Handhabung der Daten und die kürzeren Zugriffszeiten beim Lesen, da dabei keine Datenbankzugriffe mehr von Nöten sind.

Zu Beginn jedes Serverstarts wird der Cache neu initialisiert. Dies geschieht mittels zweier Initialisierungsstufen. In der ersten Stufe werden alle Ressourcenköpfe, sowie die dazugehörigen Ausprägungen aus der Datenbank geladen und in einer Collection aus allen Ressourcen gespeichert. Die einzelnen Ressourcen werden dabei in *ResourceContainer* eingehüllt. Hinter-

grund ist, dass spätere Verlinkungen von *Linked_Resources* nicht direkt auf das Objekt zeigen. Vorteil dabei ist, dass Objekte zur Laufzeit beliebig verändert oder getauscht werden können und Container beziehungsweise Verlinkungen dennoch unverändert erhalten bleiben.

Zudem ist es in der ersten Stufe wichtig, dass nicht alle Ausprägungen vollständig initialisiert werden. Bei *Linked_Resources* werden nur *ResourceIDs* geladen und noch nicht ganze Objekte. Dadurch wird eine Endlosschleife verhindert, wenn beispielsweise Ressource A mit Ressource B und andersherum verlinkt sind. In diesem Fall würde beim Laden von A, durch die Verlinkung, B geladen werden. Da B wiederum auf A verlinkt, würde das System nun erneut A initialisieren wollen. Somit würde das System diesen Prozess unendlich oft Wiederholen.

Die geladene Collection, die die *ResourceContainer* mit den jeweiligen Ressourcen beinhaltet, wird anschließen durchlaufen. Dabei wird jeder Container zu einer allgemeinen Hashmap *RESOURCES* hinzugefügt, die alle Ressourcen beinhaltet. Ebenfalls wird der Ressourcenkopf auf den gespeicherten Ressourcentyp untersucht und zusätzlich zu der passenden Hashmap des Typs hinzugefügt. Als letzter Schritt werden die Informationen, die in der Datenbank für die Ressourcentypen gespeichert sind, in die Objekte eingebracht.

Initialisierungsstufe 2 besteht dann daraus, die *ResourceIDs*, die als Verlinkungen in den Objekten gespeichert sind, mit *ResourceContainer*, in denen sich die verlinkten Objekte befinden, zu tauschen.

Eine ungefähre Idee des Ablaufes gibt Abbildung [4.3]. Dabei zu beachten ist, dass Level1 solange wiederholt wird, bis alle Ressourcenköpfe geladen sind.

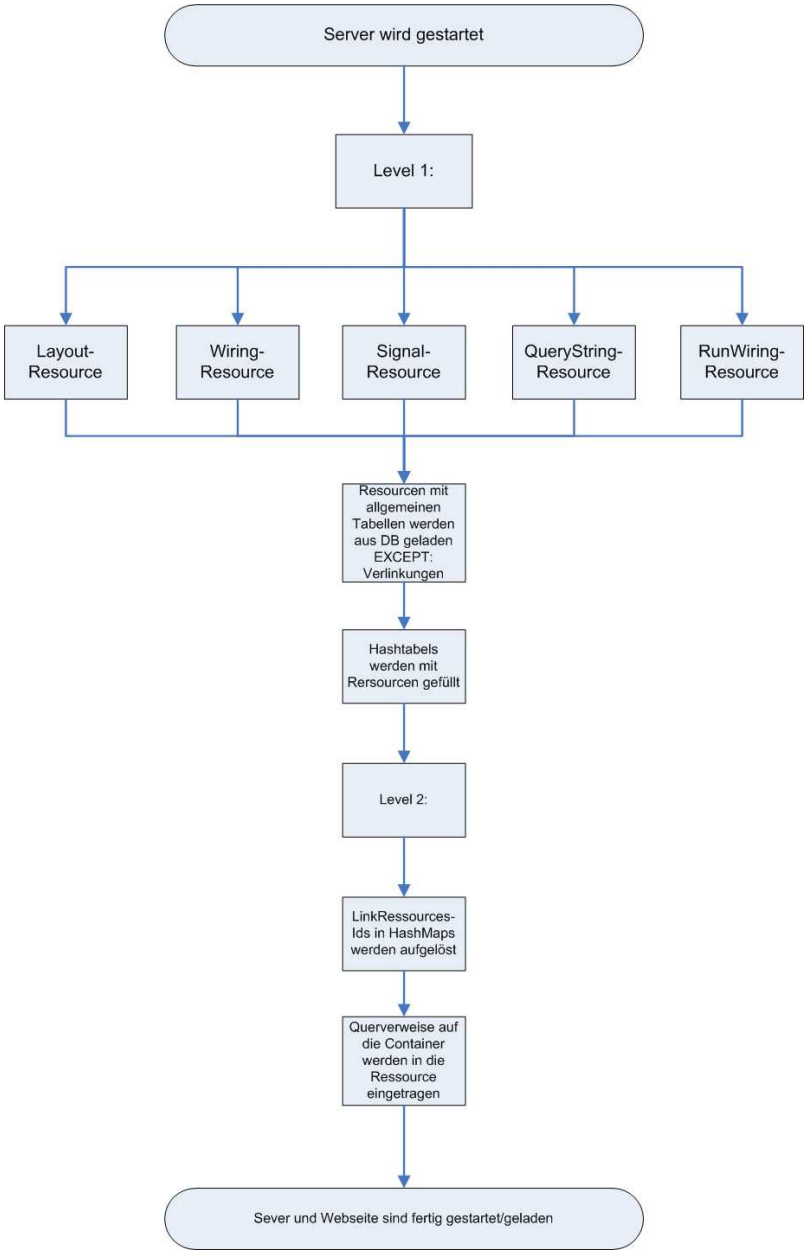

Abbildung 4.4: Ablauf Initialisierung

4.3 Implementierung

Im Folgenden wird vertieft auf die Implementierung eingegangen. Speziell wird auf den in Planung und Konzeption der Speicherung angeführten Cache eingegangen. Auch das Frontend, das in Abbildung[4.3] dargestellt ist, wird kurz erläutert. Für Methoden, die sich bei unterschiedlichen Ressourcen sehr ähneln, wird hier als Beispiel die Ressource Signal verwenden.

4.3.1 Cache Speicherung

```
1 public class ResourceProviderInitializer {
2   static ResourceMap RESOURCES = new ResourceMap();
3   static ResourceMap SIGNALS = new ResourceMap();
```

Quellcode 4.1: public class ResourceProviderInitializer{}

Die Ressourcen, die aus der Datenbank geladen werden, werden in *ResourceMaps*, wie im Code [4] zu sehen, eingeordnet. Dabei handelt es sich um eine erweiterte Klasse von linked Hashmaps, erweitert um eine *add()* Methode, die das hinzufügen vereinfacht. In den Hashmaps werden jeweils die *ResourceID* mit einem *ResourceContainer* gemappt.

```
1 public abstract class ResourceData {}
2
3 public abstract class Resource extends ResourceData {}
4
5 public class SignalResource extends Resource{}
```

Quellcode 4.2: Datenstructur von Resourcen

Bei dem *ResourceContainer* handelt es sich, wie in Planung und Konzeption der Speicherung bereits geschildert, um eine Hülle für Ressourcen. Für Ressourcen gibt es die Klasse *Resource*, die die Klasse *ResourceData* erweitert. Zudem gibt es die Klassen für Ressourcentypen, welche die Implementierung von *Resource* erweitern, um spezifische Daten zu speichern. Dadurch wird es möglich, alle Informationen des Kopfes, des Typs sowie der Ausprägungen in einem einzelnen Objekt zu speichern. Somit kann man über ein Objekt auf alle Daten einer Ressource zugreifen. Der Zugriff erfolgt über *Getter* und *Setter* Methoden.

```
1 public static void init() {
2     addAllResourcesLevel1();
3     populateLevel2();
4     populateBodyAndSort2SpecificMaps();
5   }
```

Quellcode 4.3: public static void init()

Die *init()* Methode wird bei Serverstart aufgerufen. Sie arbeitet dabei die einzelnen Schritte ab, die notwendig sind, um den Cache zu initialisieren und alle Ressourcen in ihn zu laden.

4.3.2 Initialisierungsstufe 1

```
1 private static void addAllResourcesLevel1() {
2     Collection<Resource> resources = ResourceDBInternal.↩
         getAllResources();
3     for (Resource resource : resources) {
4       addResource(resource);
5     }
6   }
```

Quellcode 4.4: public static void addAllResourcesLevel1()

Zuerst werden alle Ressourcen und dazugehörige Ausprägungen, wie *Keywords*, aus der Datenbank geladen. Anschließend werden alle Ressourcen in der Methode *addResources()* zur *ResourceMap* namens *RESOURCES* hinzugefügt. Außerdem wird geprüft, um welchen Ressourcentyp es sich handelt und dann die speziellen *ResourceMap*, zum Beispiel *SIGNALS*, erweitert.

```
1  public static Collection<Resource> getAllResources() {
2      Collection<Resource> ret = null;
3      try {
4        ret = ResourceDBImpl.getAllResources();
5      } catch (DBException e) {
6        e.printStackTrace();
7      }
8
9      addImageList(ret);
10     addFormulaList(ret);
11     addKeywordList(ret);
12     addCreatedWithIDs(ret);
13     addUsedWithIDs(ret);
14     addSeeAlsoIDs(ret);
15     addCollectionMembersIDs(ret);
16     return ret;
17   }
```

Quellcode 4.5: public static Collection<Resource> getAllResources()

Es werden beim Aufruf von *getAllResources()* alle Ressourcenköpfe aus der Datenbank geladen, wie in Zeile vier zu sehen. Ebenfalls werden alle Ausprägungen von Ressourcen über separate Funktionen geladen und der Ressource hinzugefügt.

```
1 private static void addCollectionMembersIDs(Collection<Resource> ↩
     resources) {
2   for (Resource resource : resources) {
3     List<Long> tmpCollectionMemberList = new ArrayList<>();
```

```
4       List<ResourceContainer> tmpCollectionMemberListObj = new ↩
            ArrayList<>();
5       try {
6         tmpCollectionMemberList = T_linkedResourcesDBImpl.↩
              getCollectionMembersById(resource.getResourceID());
7       } catch (DBException e) {
8         e.printStackTrace();
9       }
10      for(long memberID: tmpCollectionMemberList){
11        ResourceID memberIDObj = new ResourceID();
12        memberIDObj.setResourceID(memberID);
13        ResourceContainer memberContainer = new ResourceContainer()↩
              ;
14        memberContainer.setResource(memberIDObj);
15        tmpCollectionMemberListObj.add(memberContainer);
16      }
17      resource.setCollectionMembers(tmpCollectionMemberListObj);
18    }
19  }
```

Quellcode 4.6: private static void addCollectionMembersIDs(Collection<Resource> resources)

Wichtig dabei zu beachten ist, dass beim Laden aus der Datenbank, bei *Linked_Resources*, nur *ResourceIDs* geladen und gespeichert werden. Daher sind in der Datenbank auch nur IDs in der Tabelle gespeichert und die verlinkten Ressourcen werden in Level 2 nachgeladen.

Wie im Codeausschnitt zu sehen, wird über alle Ressourcenköpfe iteriert. Dabei werden, in dem Codebeispiel, alle *CollectionMembers* aus der Datenbank gelesen und die *ResourceID* des Members, der Ressource angehängt. Dies ist möglich, da es eine gesonderte Klasse *ResourceID* gibt, welche *Resource* erweitert und somit wie eine Ressourcentyp-Klasse gehandhabt wird. Diese Klasse beinhaltet nur eine ID vom Typen Long.

Da in *addAllResourcesLevel1()* nur die Ressourcenköpfe den spezifischen Tabellen hinzugefügt wurden, wird in *populateBodyAndSort2SpecificMaps()*, siehe [6], die Daten der Ressourcentypen aus der Datenbank und in die dazugehörige Ressource geladen.

4.3.3 Initialisierungsstufe 2

```
1  private static void populateLevel2() {
2    for (ResourceContainer resourceContainer : ResourceProvider.↩
         getAllResources()) {
3      populateLevel2(resourceContainer.getResource());
4    }
5  }
```

```
6  private static void populateLevel2(Resource resource) {
7    resource.setCreatedWith(populateLevel2(resource.getCreatedWith↩
       ()));
8    resource.setUseWith(populateLevel2(resource.getUseWith()));
9    resource.setSeeAlso(populateLevel2(resource.getSeeAlso()));
10   resource.setCollectionMembers(populateLevel2(resource.↩
       getCollectionMembers()));
11 }
12 private static List<ResourceContainer> populateLevel2(List<↩
     ResourceContainer> resourceIds) {
13   List<ResourceContainer> originalResources = new ArrayList<>();
14   for (ResourceContainer resource : resourceIds) {
15     long resourceId = resource.getResource().getResourceID();
16     originalResources.add(ResourceProvider.getResourceById(↩
         resourceId));
17   }
18   return originalResources;
19 }
```

Quellcode 4.7: private static void populateLevel2()

In Stufe 2 des Initialisierungsprozesses werden die *ResourceIDs* der *Linked_Resources*, durch komplette Objekte ersetzt. Dazu wird über alle bereits geladenen Ressourcen iteriert und die in Level 1 geladenen *ResourceIDs* ausgelesen. Anschließend werden die Ressourcen nach der *ResourceID* durchsucht. Der erhaltene *ResourceContainer* wird dann einer Liste hinzugefügt. Dies wird für alle verlinkten Ressourcen gemacht. Die entstandene Liste wird schlussendlich mit der richtigen Ressource verknüpft und gespeichert.

4.3.4 Zugriff auf Daten

Um den oben dargestellten Initialisierungsprozess so zu gestalten und später auf alle Ressourcen und Ressourcentypen aus dem Cache zugreifen zu können, benötigt man Methoden, die dies ermöglichen. Diese werden im *ResourceProvider* zur Verfügung gestellt.

```
1  public class ResourceProvider extends ResourceProviderInitializer↩
       {
2
3    public static ResourceContainer getResourceById(long resourceId↩
       ) {
4      return RESOURCES.get(resourceId);
5    }
6
7    public static Collection<ResourceContainer> getAllResources() {
8      return RESOURCES.values();
9    }
```

```
10
11   public static Collection<ResourceContainer> getAllSignals() {
12     return SIGNALS.values();
13   }
14 }
```

Quellcode 4.8: public class ResourceProvider extends ResourceProvider{}

Hier beispielhaft aufgeführt, sind die wichtigsten Methoden für den reinen Zugriff auf die erstellten *ResourceMaps*. Die erste Methode ist besonders wichtig. Zum einen für die Initialisierungsstufe 2, da dort auf spezielle Ressourcen anhand ihrer ID zugegriffen werden muss, um verlinkte Ressourcen zu laden. Zum anderen werden darüber alle Zugriffe von Usern auf ihre Daten ermöglicht. Es gibt für alle Ressourcentypen Methoden für den Zugriff.

Neben dem reinen Lesen der Daten aus dem Cache, müssen auch neue erstellt, bestehende angepasst und ungewollte gelöscht werden können. Dafür gibt es im Provider weitere Methoden, die den Zugriff regeln. Diese werden auch bei Anpassungen des Users über die im Folgekapitel aufgeführten Servlets und dessen genutzte Methoden aufgerufen.

Servlets sind Java-Klassen, die auf dem Server laufen und HTTP-Anfragen verarbeiten. Sie werden in Webanwendungen eingesetzt, um dynamische Inhalte zu generieren und mit dem Benutzer über das HTTP-Protokoll zu interagieren. Servlets können Datenbankzugriffe durchführen, Benutzereingaben validieren und Ergebnisse an den Client zurücksenden. [17]

Um die oben aufgeführte Initialisierung zu gewährleisten, sind Zugriffe auf die Datenbank notwendig. Um auf die Datenbanktabellen der Access Datenbank zuzugreifen, werden SQL-Befehle verwendet. Zudem werden prepared Statements eingesetzt, um eine SQL-Injections zu verhindern. Genauer wird darauf im Kapitel Identity- und Accessmanagement eingegangen.

4.3.5 Frontend und Servlets

Das in 4.1.1 gezeigte Frontend wurde mithilfe von JavaScript programmiert und auf einzelne Typen angepasst. Dabei wurden Servlets für Funktionalitäten wie das Anlegen oder Bearbeiten verwendet. Beispielhaft wird im Folgenden der Ablauf des Erstellens beziehungsweise des Anpassens eines *Signals* gezeigt.

```
1 <label title="Save"><input type="submit" name = "save" value="↩
    Save" class="submit-code" formaction="/labalive/signal/↩
    createupdate/"/>
```

Quellcode 4.9: Save Signal labAlive

Mit dieser Zeile wird in der umfangreichen Jsp Datei der Signal-Ressource ein Knopf implementiert, mit dem die Anfrage an das dazugehörige Servlet geleitet wird. Dabei werden in der Anfrage alle auf der Seite als Parameter definierten Werte gespeichert und können im Servlet über das HTTPRequest Objekt abgefragt werden. [18]

```
1  @WebServlet(urlPatterns = { "/signal/createupdate/"})
2  public class SignalServlet extends javax.servlet.http.HttpServlet↩
       implements javax.servlet.Servlet {
3
4      protected void doPost(HttpServletRequest request, ↩
         HttpServletResponse response){
5          doGet(request, response);
6      }
7
8      protected void doGet(HttpServletRequest request, ↩
         HttpServletResponse response){
9          User user = SessionAccess.getUser(request);
10         if (!AccessCheck.checkAccessForUser(request, user)){
11             response.sendRedirect("/labalive/auth/login/");
12             return;
13         }
14         SignalResource signalFromRequest = getSignal(request);
15         signalFromRequest.setSaveDate(LocalDate.now().toString())↩
             ;
16         signalFromRequest.setResourceType("signal");
17         if(signalFromRequest.getResourceID() >=0){
18             try{
19                 SignalDB4Servlet.updateSignal(signalFromRequest);
20             } catch (DBException e1){
21                 e1.printStackTrace();
22             }
23             sendRedirect(request, response, signalFromRequest);
24         } else {
25             try {
26                 signalFromRequest.setUserID(user.getUserId());
27                 SignalResource createdSignal = SignalDB4Servlet.↩
                     createSignal(signalFromRequest);
28                 sendRedirect(request, response, createdSignal);
29             } catch (DBException e){
30                 e.printStackTrace();
31             }
32         }
33     }
34 }
```

Quellcode 4.10: public class SignalServlet{}

In Zeile eins wird definiert, unter welcher URL das Servlet zu erreichen ist. Dabei fällt auf,

dass im Code [2] noch ein */labalive* davorsteht. Der Grund dafür ist, dass man auf der Webseite noch den Kontext, unter dem die Webseite arbeitet, angeben muss, da ansonsten das Servlet nicht gefunden werden kann. Denn unter diesem Kontext arbeitet die gesamte Webseite.

Danach wird definiert, dass alle Anfragen von der *doGet()* Methode verarbeitet werden. Darin wird zunächst ein Signal aus den übergebenen Parametern der Anfrage erstellt. Da die Parameter auf der Webseite definiert sind und vom User eingetragen werden, können diese ohne Probleme aus der Anfrage herausgelesen werden. Dies ist in *getSignal()* umgesetzt.

Zuerst wird geprüft, ob der Zugriff auf diese Ressource von dem User erlaubt ist. Falls nicht, wird er zur Loginseite weitergeleitet. Beim neuen Anlegen einer Ressource gibt die Funktion immer *true* zurück.

Folgernd werden allgemeine Werte, wie das aktuelle Datum, gesetzt. Zuletzt wird überprüft, ob das übergebene Signal neu angelegt werden muss oder ob es bereits besteht und der User Änderungen vorgenommen hat. Dies wird mithilfe der *ResourceID* gemacht. Da diese auf der Webseite nur bei bereits bestehenden Ressourcen gesetzt ist, kann man anhand dessen prüfen, welche Methode aufgerufen werden muss. Wird die Ressource neu erzeugt, wird eine *UserID* gesetzt.

Über die Methoden *updateSignal()* und *createSignal()* wird sichergestellt, dass das neue oder abgeänderte Signal in der Datenbank persistent geändert oder abgelegt wird. Ebenfalls werden darin die bereits vorgestellten Methoden, die der *ResourceProvider* zur Verfügung stellt, aufgerufen, um den Cache aktuell zu halten.

5 Access- und Identity Management für labAlive

Accessmanagement bezieht sich auf die Verwaltung und Kontrolle von Benutzerzugriffen auf Systeme, Ressourcen und Daten. Es umfasst die Gewährung, Überwachung und Einschränkung von Zugriffsrechten, um die Informationssicherheit zu gewährleisten und Richtlinien einzuhalten. [18]

Identitymanagement bezieht sich auf die Verwaltung von Identitäten und Zugriffsrechten von Benutzern. Es umfasst die Erfassung, Verwaltung und Verifizierung von Identitätsinformationen, um den Benutzerzugriff effektiv zu steuern und Sicherheitsrisiken zu minimieren. [18]

5.1 Planung und Konzeption

5.1.1 Login und Authentifizierung

Neben einem neuen Frontend des Logins, welches zunächst hauptsächlich funktional sein sollte, musste das Backend angepasst werden. Durch das Verwenden einer Nutzername-Passwort-Kombination bedarf es neuer Eingabefelder, neuer Funktionen im Servlet sowie neuer Datenbankzugriffe. Der Grundbaustein war bereits ausgearbeitet. Dazu zählen die Funktionalität des Servlets, die Verarbeitung des Tokens aber auch Teile der Datenbankzugriffe.

Wie in [3.2] bereits gesehen, wurde das Token zum Login verwendet. Es wurde bei der Registrierung erstellt. Ebenfalls wurde eine Session und auf Wunsch des Users ein permanenter Cookie erstellt und in einer Antwort an den Client mitgeschickt.

Eine Web-Sitzung(*Session*), auch als Sitzungszustand oder Sitzungsverwaltung bezeichnet, bezieht sich auf den Zustand der Interaktion eines Benutzers mit einer Website über einen bestimmten Zeitraum hinweg. Während einer Web-Sitzung können Benutzeraktionen, wie das Anzeigen von Seiten, das Ausfüllen von Formularen oder das Speichern von Daten, verfolgt und verwaltet werden. Dies ermöglicht es der Website, den Zustand und die Informationen des

Benutzers zwischen verschiedenen Seiten oder Anfragen beizubehalten. [19]

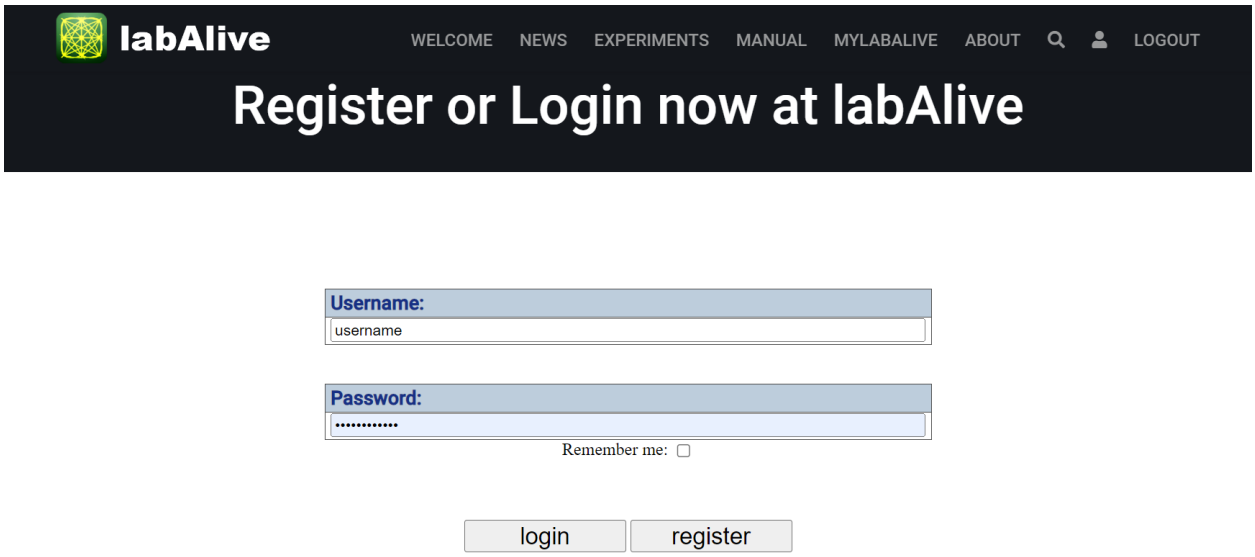

Abbildung 5.1: Neues Login labAlive

Das neue Konzept ist in [5.1] zu sehen. Die Idee ist es, das Login-Konzept so zu überarbeiten, dass Nutzer ihren Nutzernamen und ihr Passwort problemlos im Browser speichern können. Damit andere Funktionen, wie der permanente Cookie, nicht verloren gehen, wird im Hintergrund immer zusätzlich ein Token bei der Registrierung erstellt. Dieses wird für den Wert, der in den Cookie *user* geschrieben wird. Der Cookie dient dazu, dass der User nach Ablauf seiner Session automatisch wieder eingeloggt wird. Durch das Token können aus dem Cookie keine persönlichen Informationen ausgelesen werden und jeder Nutzer hat zur automatischen Authentifizierung sein individuelles Token.

Die eingegebenen Zugangsdaten des Users werden an ein Servlet übermittelt, wo sie überprüft werden. Dabei wird beim Login geprüft, ob die eingegebenen Informationen korrekt sind. Wenn sich ein Nutzer registrieren will, wird geprüft, ob der Nutzername bereits vorhanden ist. Falls der Username noch nicht vorhanden ist, wird ein Token generiert und dieser mit den Logindaten in einer Tabelle der Datenbank gespeichert. Dabei wird das Passwort nicht als Klartext, sondern als Hashwert, mit zusätzlichem Salt gespeichert. Anschließend werden Username, Token und die von der Datenbank erzeugte *UserID* in einem Objekt gespeichert. Es wird auch eine neue Session initialisiert, in der dieses Objekt gespeichert wird. Dadurch ist ein User eindeutig identifizierbar.

5.1.2 Rollenmanagement und Autorisierung

Zusätzlich zu den Login- und Identifikations-Mechanismen, sollen Rollen für User und den Administrator eingeführt werden. Dazu wird das Objekt, in dem die Nutzerdaten bei Überprüfungen aus der Session gespeichert werden, um eine Rolle erweitert. Ebenfalls werden diese Rollen in der Datenbank jedem Nutzer zugeordnet.

Die festgelegten Rollen sind Admin, Moderator und User. Dabei wurden bis zum jetzigen Zeitpunkt noch keine Einschränkungen oder Rechte für die einzelnen Rollen festgelegt. Lediglich die Funktionalität um dies zu Verwalten oder Rechte eines Users anhand seiner Rolle zu überprüfen. Die genauen Rechte müssen noch umgesetzt werden.

Doch allein mit Rollen ist der autorisierte Zugriff auf persönliche Daten nicht gewährleistet. Durch die Rollen werden allgemeine Gruppen festgelegt, mit Rechten und Einschränkungen. Um auf persönliche Ressourcen zugreifen zu können, wird beim Anlegen von Ressourcen, die *UserID* des/der Erstellers/Erstellerin in der Datenbank und im Cache gespeichert. Beim Zugriff eines Users auf seine Daten, werden alle Ressourcen durchsucht, und die gespeicherte *UserID* von einzelnen Ressourcen mit der ID, die in der Session gespeichert ist, verglichen. Somit können alle Ressourcen von einem User ausfindig gemacht und diesem auf der Webseite angezeigt werden.

5.2 Maßnahmen gegen Cyberangriffe

In diesem Abschnitt werden mögliche Angriffe auf das Identity- und Accessmanagement und Maßnahmen gegen diese beleuchtet. Das Wichtigste bei der Umsetzung ist, dass die Logindaten der User sicher verarbeitet und gespeichert werden. Um das Passwort zu sichern, wird es vor der Speicherung in die Datenbank, gehasht und mit einem Salt Wert zusätzlich erweitert.

Das Hashen und Salzen von Passwörtern ist ein wichtiger Schritt, um die Sicherheit von Benutzerkonten zu gewährleisten. Beim Hashen wird das Passwort in einen eindeutigen Hashwert umgewandelt, der nicht zurück in das ursprüngliche Passwort umgewandelt werden kann. Dadurch wird verhindert, dass im Falle eines Datenlecks oder eines erfolgreichen Angriffs die Passwörter in Klartextform abgegriffen werden. [20]

Das Salzen ist ein weiterer Schritt, bei dem zu jedem Passwort eine zufällige Zeichenkette (*Salt*) hinzugefügt wird, bevor es gehasht wird. Durch die Verwendung von Salt wird sichergestellt, dass selbst bei Verwendung desselben Passworts durch verschiedene Benutzer unterschiedliche Hashwerte erzeugt werden. Dies erschwert Angreifern den Einsatz vorberechneter Tabellen (*Rainbow Tables*), um den Hashwert zurückzuführen und das ursprüngliche Passwort zu ermitteln. [20]

Um das umzusetzten, wird der Hash-Algorithmus *BCrypt* zum Verschlüsseln sowie für die Passwortüberprüfung beim Login verwendet. *BCrypt* ist ein speziell für die Passwortverschlüsselung entwickelter Hash-Algorithmus. *BCrypt* verwendet individuelle Salt-Werte für jeden Hash-Vorgang und führt mehrere Runden des Hashings durch, um Brute-Force-Angriffe zu erschweren. Das bedeutet, dass der Aufwand, ein gehashtes Passwort zu knacken, mit der verfügbaren Rechenleistung exponentiell ansteigt. [21]

Durch die Kombination von Hashing und Salzen mit *BCrypt* wird die Sicherheit der Passwörter erheblich gestärkt. Selbst wenn ein Angreifer Zugriff auf die gehashten Passwörter erhält, ist es äußerst schwierig, die ursprünglichen Passwörter wiederherzustellen.

Ebenfalls wichtig zu verhindern sind SQL-Injections. Dabei handelt es sich um eine Sicherheitslücke, bei der schädlicher SQL-Code in eine webbasierte Anwendung eingeschleust wird. Dadurch kann ein Angreifer auf unerlaubte Weise auf die zugrunde liegende Datenbank zugreifen, Daten manipulieren oder das gesamte System übernehmen.

```
String sql = "SELECT * FROM users WHERE username = ? AND password↩
    = ?";
PreparedStatement preparedStatement = connection.prepareStatement↩
   (sql);
preparedStatement.setString(1, "username");
preparedStatement.setString(2, "secret123");
```

Quellcode 5.1: Besipiel Prepared Statement

Um dies grundlegend zu verhindern, werden in der Datenbank prepared Statements verwendet. Siehe [5]. Dabei handelt es sich um Platzhalter bei SQL-Anweisungen. Die Anwendung bindet die tatsächlichen Werte an die Platzhalter und sendet sie separat an die Datenbank.

Für die allgemeine Sicherheit und den Datenschutz wird als Cookie-Wert lediglich das erzeugte Token verwendet. Beim Login wird geprüft, ob das Token des Cookies mit dem Token in der Datenbank übereinstimmt. Somit geschieht die User Verwaltung lediglich über das zugewiesene Token.

Da bei der Webseite, eine HTTPS-Verbindung genutzt wird, ist die Kommunikation zwischen Client und Server ebenfalls abgesichert und beispielsweise gegen Session-Hijacking gesichert. [22] Session-Hijacking ist eine Angriffstechnik, bei der versucht wird, die Kontrolle über eine laufende Sitzung eines authentifizierten Benutzers zu übernehmen, um unberechtigten Zugriff auf die Ressourcen und Privilegien des Benutzers zu erlangen. [22] Da die Kommunikation abgesichert ist, ist dies grundlegend abgesichert.

Bei allen angeführten Punkten, handelt es sich um Grundkonzepte, die keine vollständige Sicherheit bieten.

5.3 Implementierung

```
<th style="vertical-align:middle; font-size:18px"> Password:</th>
  <tr>
    <td><input type="password" name="p" autocomplete="current-↩
       password" size="70" style="font-size:14px;" placeholder="↩
       password" required></td>
  </tr>
```

Quellcode 5.2: Frontend Eingabefelder

Im Frontend gibt es Eingabefelder für Username und Passwort. Beim Passwort wurde der Type Parameter auf *password* gesetzt. Das sorgt beim User dafür, dass das Passwort beim Eintippen nicht als Klartext, sondern als Punkte angezeigt wird. [23] Dadurch wird die Privatsphäre erhöht,

da keiner zufällig oder gezielt das Passwort am Bildschirm mitlesen kann.

Damit Passwort und Username vom Browser als diese erkannt werden, wird bei beiden Feldern, beispielhaft beim Passwort zu sehen, die Option *autocomplete* hinzugefügt. Durch die richtigen Parameter hinter diesem Wert fragt der Browser beim Registrieren, ob die Logindaten gespeichert werden sollen. Beim Login füllt er die Felder automatisch mit den gespeicherten Werten aus. [23]

```
1  Token token = createNewToken();
2  HashMap<String, String> credentials = getenteredCredentials(↩
       request);
3      try {
4          if (credentials.get("username") == null){
5              return RegistrationResult.FAILED;
6          }
7      User user = DBUser.createUser(credentials.get("username"), ↩
           token, credentials.get("passwort"));
8      LoginLogic.getInstance().setUserInSession(request, response, ↩
           user);
9      LoginServlet.processRememberme(request, response);
10     return RegistrationLogic.RegistrationResult.SUCCESS;
```

Quellcode 5.3: Nutzer Registrieren

Beim Drücken auf den Registrieren-Button wird das dazugehörige Servlet aufgerufen. Darin wird überprüft, ob es den Username bereits gibt. Falls nicht, werden sich, wie in [11] zu sehen, die Login-Informationen aus der Anfrage geladen. Danach werden diese, mit der Methode *createUser()*, in die Datenbank geschrieben. Beim Holen der Eingabedaten wird das Passwort direkt gehasht und mit einem Salt erweitert. Siehe [3].

Ebenfalls wird eine neue Session erzeugt und mit *processRememberme()* geprüft, ob der Nutzer einen persistenten Cookie haben will oder nicht. Falls ja, wird dieser in derselben Methode gesetzt und anschließend an den User übermittelt.

```
1  String salt = BCrypt.gensalt();
2  String hashedPassword = BCrypt.hashpw(password, salt);
```

Quellcode 5.4: Hashen des Passworts

```
1  Token token = getTokenFromCookie(request);
2  HashMap<String, String> credentials = LoginLogic.↩
       getenteredCredentials(request);
3  User alreadyLoggedinUser = SessionAccess.getUser(request);
```

```
4  {...}
5  if (alreadyLoggedinUser == null & credentials.get("passwort") != ↩
       null) {
6      String correctPassword = null;
7      try{
8          correctPassword = DBUser.getUserpassword(credentials.get(↩
               "username"));
9          if (correctPassword == null){
10             return LoginResult.FAILED;
11         }
12     }catch (Exception e){
13         e.printStackTrace();
14     }
15     if (LoginLogic.checkIfPasswordIsCorrect(credentials.get("↩
           passwort"), correctPassword) ) {
16         try {
17             setUserInSession(request, response, DBUser.↩
                   getUserByUsername(credentials.get("username")));
18         } catch (DBException e) {
19             throw new RuntimeException(e);
20         }
21         return LoginResult.SUCCESS; // show admin page
22     }
23 }
```

Quellcode 5.5: Login Überprüfung

Durch das Drücken auf den Login Button im Frontend wird über das Servlet die Methode *login()* aufgerufen. Darin werden sich aus der Anfrage die wichtigsten Parameter, wie der eventuell vorhandene Cookie und die eingegebenen Informationen, geholt. Danach werden verschiedene Kombinationen des Cookies, der Logininformationen und der Session geprüft. In [24] ist die wichtigste Abfrage gezeigt. Diese trifft ein, wenn der User keine Session und kein Cookie hat. In diesem Fall wird geschaut, ob es den Username in der Datenbank bereits gibt. Falls ja, wird das gehashte Passwort aus der Datenbank, mit dem eingegebenen Passwort verglichen. Sind die Eingabedaten falsch, wird dies dem User auf der Webseite angezeigt und er kann es erneut versuchen.

```
1 LoginLogic.deletePersistentCookie(request, response);
2 request.getSession().invalidate();
3 forward("/content/auth/logout/loggedout.jsp", request, response);
```

Quellcode 5.6: Logout Dynamik

Der LogoutButton wird im allgemein verwendeten Header eingebettet, damit man sich auf jeder Seite, zu jeder Zeit ausloggen kann. Dabei wird man zum dazugehörigen Servlet geleitet. Darin wird die Session invalidiert und falls vorhanden wird der gesetzte Cookie mit dem Token gelöscht.

```
1 public static Collection<ResourceContainer> getAllWirings(↩
     HttpServletRequest request) {
2   Collection<ResourceContainer> wirings = filterForUser(WIRINGS.↩
       values(), request);
3   return wirings;
4 }
```

Quellcode 5.7: public static Collection<ResourceContainer> getAllWirings(HttpServletRequest request)

Neben dem Login und der Registrierung musste auch der Zugriff auf Ressourcen geprüft werden. Da auf Ressourcen nur über den *ResourceProvider*, also über den Cache, zugegriffen wird, musste man diesen Zugriff überprüfen. Dazu werden in der Methode *filterForUser()*, im obigen Beispiel, alle *Wirings* aus der Hashmap auf Gleichheit ihrer gespeicherten *UserID*, mit der ID des Users, der auf die Ressourcen zugreifen will, geprüft. Somit werden nur die Ressourcen zurückgegeben, die dem User gehören, der angefragt hat.
Schlussendlich wurden die Anfänge für das Rollenmanagement implementiert. Dazu zählt das Speichern einer Rolle für jeden Benutzer und die grundlegende Überprüfung, ob ein User eine Aktion ausführen darf. Da ein Admin alle Aktionen durchführen darf, wird dies überprüft. Die genauen Privilegien und Rechte müssen noch implementiert werden.

6 Ausblick und Weiterentwicklung

Während der Projekt- und Bachelorarbeit, wurden neben den implementierten Teilen noch weitere Ideen ausgearbeitet, welche in Zukunft die labAlive Webseite komfortabler und sicherer machen könnten.

Dazu gehört das Teilen von Ressourcen mit anderen Nutzern oder der gesamten Webseite. Beim Teilen mit der gesamten Webseite geht es um das Veröffentlichen von einzelnen Ressourcen, auf die dann jeder User der Webseite über eine gesonderte Seite Zugriff hat. Veröffentlichte Ressourcen sollen von allen Usern angeschaut und ausgeführt werden können. Das Bearbeiten ist dabei nicht vorgesehen.

Der Ablauf soll folgendermaßen aussehen. Beim Anlegen oder Speichern einer Ressource kann der User sagen, zum Beispiel durch eine Checkbox, dass er seine erstellte Ressource gerne veröffentlichen würde. Bei dieser Ressource wird der *publishedStatus*, welcher Teil jedes Objekts ist, auf beispielsweise „proposed“ gesetzt. Der Admin oder auch Moderator, sind dann in der Lage, Ressourcen zu sehen, die als vorgeschlagen markiert sind. Sie können dann überprüfen, ob das Veröffentlichen sinnvoll ist.

Wenn die Ressource in Ordnung ist und veröffentlicht werden soll, wird sie komplett kopiert, und als Ressource desjenigen gespeichert, der sie überprüft hat und freigeben möchte. Der *publishedStatus* der ursprünglichen Ressource wird auf „published“ gesetzt, sodass der Nutzer diese nicht erneut vorschlagen kann und eine Rückmeldung erhält, dass seine Ressource veröffentlicht wurde. Am Ende wird die kopierte Ressource veröffentlicht und auf einer gesonderten Seite zur Verfügung gestellt.

Wird die Ressource abgelehnt, egal aus welchen Gründen, wird der *publishedStatus* auf beispielsweise „declined“ gesetzt. Das stellt sicher, dass der Nutzer dieselbe Ressource nicht noch einmal vorschlagen kann. Folglich wird die Ressource auch nicht kopiert und nicht veröffentlicht.

Da der *publishedStatus* in der Datenbank als integer gespeichert werden soll, müsste man ein Mapping von verschiedenen Status auf eine Zahl machen. Eine mögliche Zuordnung ist in [6.1]

zu sehen.

Status	Integer	Erläuterung
default	0	Jede erstellte Ressource erhält diesen Wert
proposed	1	Dieser Wert wird gesetzt, wenn ein User eine Ressource zum Veröffentlichen vorschlägt
declined	2	Das Veröffentlichen wurde vom ADMIN oder MODERATOR abgelehnt und kann nicht erneut vorgeschlagen werden
published	3	Die Ressource wurde kopiert und ist veröffentlicht

Tabelle 6.1: Mögliches Mapping von *publishedStatus* auf Integer

Neben dem Veröffentlichen sollen User in der Lage sein, Ressourcen nicht mit allen, sondern nur mit gewissen Personen zu teilen. Dazu wurde die Idee von befristeten Links ausgearbeitet. Dabei soll ein User in der Lage sein, einen Link zu erstellen, über den andere Nutzer, auf eine seiner privaten Ressourcen zuzugreifen. Eine mögliche Umsetzung wäre, dass der Link auf eine *ResourceID* linkt, aber zusätzlich einen Parameter beinhaltet, der beim Erstellen des Links gespeichert wird. Beim Aufrufen wird dann der gespeicherte Parameter mit dem Parameter im Link verglichen. Stimmen diese überein kann man darauf zugreifen. So wäre es möglich einen Link für eine bestimmte Ressource an andere zu verteilen, mit welchem sie auf die Ressource zugreifen können. Der Parameter könnte zum Beispiel eine jedes Mal neu generierte *UUID* sein.

Den Link könnte man mit weiteren Parametern erweitern, die beispielsweise die Rechte desjenigen Users, der über den Link auf die Ressource zugreift, einschränken oder erweitern. So wäre es möglich, gemeinsam eine Ressource zu bearbeiten, aber auch eine Ressource nur zum Lesen oder Ausführen zu teilen.

7 Fazit

Zusammenfassend lässt sich sagen, dass diese Bachelorarbeit einen Beitrag zur Weiterentwicklung und Stärkung der Websecurity des labAlive Identity- und Access Managementsystems geleistet hat. Durch die Überarbeitung des Login-Verfahrens und die Implementierung grundlegender Standards sowohl in Bezug auf Komfort als auch Sicherheit wurde eine solide Basis geschaffen. Zudem wurde das Datenmanagement verbessert, wodurch die Effizienz und Sicherheit der Plattform erhöht wurden.

Die erzielten Fortschritte ermöglichen es den Usern von labAlive, ihre Projekte effizienter und sicherer umzusetzen, was den Mehrwert und die Zuverlässigkeit der Plattform steigern kann. Zusätzlich wird die Sicherheit sensibler Daten gewährleistet und potenzielle Sicherheitsrisiken minimiert.

Zukünftige Erweiterungen und Verbesserungen könnten darauf abzielen, die Benutzerfreundlichkeit der Plattform weiter zu optimieren und zusätzliche Sicherheitsmaßnahmen zu implementieren, um den ständig wachsenden Bedrohungen im Bereich der Cybersecurity gerecht zu werden. Es ist anzuerkennen, dass die Weiterentwicklung des Identity- und Accessmanagement-Systems und die Sicherung der Webressourcen eine fortlaufende Aufgabe sind, um den aktuellen und zukünftigen Anforderungen gerecht zu werden.

Insgesamt hat diese Bachelorarbeit gezeigt, dass eine ganzheitliche Herangehensweise an Identity- und Access Management sowie Websecurity essenziell ist, um den Schutz sensibler Daten und die Gewährleistung einer vertrauenswürdigen Online-Plattform zu gewährleisten. Die erzielten Ergebnisse und Erkenntnisse bieten eine solide Grundlage für weitere Entwicklungen in diesem Bereich.

8 Anhang

Im Anhang ist eine umfassende Darstellung der Funktionen, die im Hauptteil des Dokuments nur in Ausschnitten gezeigt wurden, enthalten. Dadurch wird die Vollständigkeit des Themas gewährleistet. Auch hier wird nur die Ressource *Signal* aufgeführt, da sich die Methoden aller Ressourcen ähneln.

```
1  @WebServlet(urlPatterns = { "/signal/createupdate/"})
2  public class SignalServlet extends javax.servlet.http.HttpServlet
       implements javax.servlet.Servlet {
3      private static final long serialVersionUID = -469487547196518
           2952L;
4
5      protected void doPost(HttpServletRequest request,
           HttpServletResponse response)throws ServletException,
           IOException{
6          doGet(request, response);
7      }
8      protected void doGet(HttpServletRequest request,
           HttpServletResponse response)throws ServletException,
           IOException{
9          User user = SessionAccess.getUser(request);
10         if (!AccessCheck.checkAccessForUser(request, user)){
11             response.sendRedirect("/labalive/auth/login/");
12             return;
13         }
14         SignalResource signalFromRequest = getSignal(request);
15         signalFromRequest.setSaveDate(LocalDate.now().toString())
               ;
16         signalFromRequest.setResourceType("signal");
17         if(signalFromRequest.getResourceID() >=0){
18             try{
19                 SignalDB4Servlet.updateSignal(signalFromRequest);
20             } catch (DBException e1){
21                 e1.printStackTrace();
22             }
23             sendRedirect(request, response, signalFromRequest);
24         } else {
25             try {
26                 signalFromRequest.setUserID(user.getUserId());
27                 SignalResource createdSignal = SignalDB4Servlet.
                       createSignal(signalFromRequest);
28                 sendRedirect(request, response, createdSignal);
29             } catch (DBException e){
30                 e.printStackTrace();
31             }
32         }
33     }
34     private void sendRedirect(HttpServletRequest request,
           HttpServletResponse response, SignalResource signal) {
35         try {
```

```
36              response.sendRedirect(request.getContextPath() + "/↩
                    resource/signal.jsp?resourceID=" + signal.↩
                    getResourceID());
37          } catch (IOException e) {
38              e.printStackTrace();
39          }
40      }
41      private SignalResource getSignal(HttpServletRequest request){
42          SignalResource signal = new SignalResource();
43
44          String resourceID_str = request.getParameter("resourceID"↩
                );
45          double power = Double.parseDouble(request.getParameter("↩
                power"));
46          double peakValue = Double.parseDouble(request.↩
                getParameter("peakValue"));
47          long numberOfSamples = Long.parseLong(request.↩
                getParameter("numberOfSamples"));
48          Format format = Format.getFormat(request.getParameter("↩
                format"));
49          FileSize fileSize = new FileSize(Integer.parseInt(request↩
                .getParameter("fileSize")));
50          double lenghInSec = Double.parseDouble(request.↩
                getParameter("lengthInSec"));
51          double sampleRate = Double.parseDouble(request.↩
                getParameter("sampleRate"));
52
53          if(!TextUtils.isEmpty(resourceID_str)){
54              long resourceID = Long.parseLong(resourceID_str);
55              signal.setResourceID(resourceID);
56          }
57          signal.setPower(power);
58          signal.setPeakValue(peakValue);
59          signal.setNumberOfSamples(numberOfSamples);
60          signal.setFormat(format);
61          signal.setFileSize(fileSize);
62          signal.setLenghInSec(lenghInSec);
63          signal.setSampleRate(sampleRate);
64          return signal;
65
66      }
```

Quellcode 8.1: Klasse SignalServlet

```
1  public class SignalSelector4Jsp {
2      static int init = 0;
3      public static Collection<ResourceContainer> getAllSignals() {
4          if(init == 0){
5              ResourceProviderInitializer.init();
```

```
6               init = 1;
7           }
8           return ResourceProvider.getAllSignals();
9       }
10  }
```

Quellcode 8.2: Klasse SignalSelector4Jsp

```
1   @WebServlet(urlPatterns = { "/signal/delete/"})
2   public class SignalServlet4Delete extends javax.servlet.http.↩
        HttpServlet implements javax.servlet.Servlet{
3       private static final long serialVersionUID = -469487547196518↩
            2952L;
4
5       protected void doPost(HttpServletRequest request, ↩
            HttpServletResponse response)throws ServletException, ↩
            IOException{
6           doGet(request, response);
7       }
8       protected void doGet(HttpServletRequest request, ↩
            HttpServletResponse response)throws ServletException, ↩
            IOException{
9           User user = SessionAccess.getUser(request);
10          if (!AccessCheck.checkAccessForUser(request, user)){
11              response.sendRedirect("/labalive/auth/login/");
12              return;
13          }
14          SignalResource signalFromRequest = getSignal(request);
15          try{
16              signalFromRequest.setResourceType("signal");
17              SignalDB4Servlet.deleteSignal(signalFromRequest);
18              sendRedirect(request, response, signalFromRequest);
19          } catch (Exception e){
20              e.printStackTrace();
21          }
22      }
23      private void sendRedirect(HttpServletRequest request, ↩
            HttpServletResponse response, SignalResource signal) {
24          try {
25              response.sendRedirect(request.getContextPath() + "/↩
                    resource/signal.jsp");
26          } catch (IOException e) {
27              e.printStackTrace();
28          }
29      }
30      private SignalResource getSignal(HttpServletRequest request){
31          SignalResource signal = new SignalResource();
32
33          long resourceID = Long.parseLong(request.getParameter("↩
```

```
            resourceID"));
34          signal.setResourceID(resourceID);
35
36          return signal;
37      }
```

Quellcode 8.3: Klasse SignalServlet4Delete

```
1  public class SignalDB4Servlet {
2      public static Collection<ResourceContainer> getAllSignals() {
3          return ResourceProvider.getAllSignals();
4      }
5      public static SignalResource getSignalsByResourceId(long ↩
           resourceId){
6          return (SignalResource) ResourceProvider.getResourceById(↩
               resourceId).getResource();
7      }
8      public static SignalResource createSignal(SignalResource ↩
           signal) throws DBException {
9          signal = (SignalResource) ResourceDBInternal.↩
               createResource(signal); // resourceId is set
10         signal = R_SignalDBImpl.createSignal(signal); // version ↩
               is set / might be changed
11         return signal;
12     }
13     public static void updateSignal(SignalResource signal) throws↩
           DBException {
14         R_SignalDBImpl.updateSignal(signal);
15         ResourceProvider.updateResource(signal);
16     }
17     public static void deleteSignal(SignalResource signal) throws↩
           DBException {
18         R_SignalDBImpl.deleteSignal(signal);
19         ResourceDBInternal.deleteResource(signal);
20     }
21 }
```

Quellcode 8.4: Klasse SignalDB4Servlet

```
1  public class R_SignalDBImpl {
2      public static SignalResource populateSignal(SignalResource ↩
           signalResource) throws DBException {
3          DBSelectCommand cmd = getSignalByResourceID(↩
               signalResource);
4          return (SignalResource) cmd.run();
5      }
6      private static DBSelectCommand getSignalByResourceID(↩
           SignalResource signal) throws DBException {
7          String preparedStat = "SELECT * FROM R_Signal where ↩
               ResourceID = ?";
```

```
8           Object[] parameterList = {signal.getResourceID()};
9           System.out.println(signal.getResourceID());
10          DBSelectCommand cmd = new DBSelectCommand(preparedStat, ↩
                parameterList) {
11              protected Object handleResultSet(ResultSet rs) throws↩
                    Exception {
12                  while (rs.next()) {
13                      signal.setSignalID(rs.getLong("signalID"));
14                      signal.setPower(rs.getDouble("power"));
15                      signal.setPeakValue(rs.getDouble("peakValue")↩
                            );
16                      signal.setNumberOfSamples(rs.getLong("↩
                            numberOfSamples"));
17                      signal.setFormat(Format.getFormat(rs.↩
                            getString("format")));
18                      signal.setFileSize(new FileSize(rs.getInt("↩
                            fileSize")));
19                      signal.setLenghInSec(rs.getDouble("↩
                            lengthInSec"));
20                      signal.setSampleRate(rs.getDouble("sampleRate↩
                            "));
21                  }
22                  return signal;
23              }
24          };
25          return cmd;
26      }
27
28
29      public static void updateSignal(SignalResource signal) throws↩
            DBException {
30          long ResourceId = signal.getResourceID();
31          double newPower = signal.getPower();
32          double newPeakValue = signal.getPeakValue();
33          long newNumberOfSamples = signal.getNumberOfSamples();
34          Format newFormat = signal.getFormat();
35          FileSize newFileSize = signal.getFileSize();
36          double newLenghInSec = signal.getLenghInSec();
37          double newSampleRate = signal.getSampleRate();
38          String preparedStat = "UPDATE R_Signal SET Power = ? , ↩
                PeakValue = ? , NumberOfSamples = ? , format = ?, ↩
                FileSize = ?, LengthInSec = ?, SampleRate = ? WHERE ↩
                ResourceID = ?";
39          Object[] parameterList = {newPower, newPeakValue, ↩
                newNumberOfSamples, newFormat, newFileSize, ↩
                newLenghInSec, newSampleRate, ResourceId};
40          DBUpdateCommand cmd = new DBUpdateCommand(preparedStat, ↩
                parameterList);
41          cmd.run();
```

```
42      }
43
44      public static Integer deleteSignal(SignalResource signal) ↩
           throws DBException {
45          String selectQuery = "DELETE FROM R_Signal WHERE ↩
               ResourceID = ?";
46          Object[] parameterList = {signal.getResourceID()};
47          System.out.println(selectQuery);
48          DBUpdateCommand cmd = new DBUpdateCommand(selectQuery, ↩
               parameterList);
49          return (Integer) cmd.run();
50      }
51
52      public static SignalResource createSignal(SignalResource ↩
           signal) throws DBException {
53          String preparedStat = "INSERT INTO R_Signal (ResourceID, ↩
               Power, PeakValue, NumberOfSamples, format, FileSize, ↩
               LengthInSec, SampleRate) VALUES ("
54                  + "? , "//ResourceID
55                  + "? , "//Power
56                  + "? , "//PeakValue
57                  + "? , "//NumberOfSamples
58                  + "? , "//Format
59                  + "? , "//FileSize
60                  + "? , "//LengthInSec
61                  + "? )";//SampleRate
62          Object[] parameterList = {
63                  signal.getResourceID(),
64                  signal.getPower(),
65                  signal.getPeakValue(),
66                  signal.getNumberOfSamples(),
67                  signal.getFormat().getFormatString(),
68                  signal.getFileSize().getSizeInByte(),
69                  signal.getLenghInSec(),
70                  signal.getSampleRate()};
71          DBUpdateCommand cmd = new DBUpdateCommand(preparedStat, ↩
               parameterList);
72          System.out.println(preparedStat);
73          cmd.run();
74          long signalId =getSignalIdByResourceID(signal.↩
               getResourceID());
75          signal.setSignalID(signalId);
76          return signal;
77      }
78
79      private static long getSignalIdByResourceID(long resourceId) ↩
           throws DBException {
80          String preparedStat = "SELECT * FROM R_Signal where ↩
               ResourceID = ?";
```

```
81          Object[] parameterList = {resourceId};
82          DBSelectCommand cmd = new DBSelectCommand(preparedStat, ↩
               parameterList) {
83              protected Object handleResultSet(ResultSet rs) throws↩
                    SQLException {
84                  rs.next();
85                  return rs.getLong("SignalID");
86              }
87          };
88          return (Long) cmd.run();
89      }
90  }
```

Quellcode 8.5: Klasse R_SignalDBImpl

```
1   @WebServlet(urlPatterns = { "/auth/login/" })
2   public class LoginServlet extends BaseServlet {
3       private static final long serialVersionUID = -786677937212819↩
           0736L;
4
5       @Override
6       protected void dispatch(HttpServletRequest request, ↩
           HttpServletResponse response) throws ServletException, ↩
           IOException {
7           RequestContext requestContext = RequestContext.instance(↩
               request);
8           requestContext.setDirectory(Directory.AUTH);
9           LoginLogic.LoginResult result = LoginLogic.getInstance().↩
               login(request, response); // TokenFromRequestParameter↩
                Or Cookie
10          if(result.equals(LoginLogic.LoginResult.↩
               ALREADY_LOGGED_IN_WITH_TOKEN)) {
11              User user = new User();
12              try{
13                  user = DBUser.loadUser(LoginLogic.↩
                       getTokenFromCookie(request));
14                  SessionAccess.setUser(request, user);
15              }catch (Exception e){
16                  e.printStackTrace();
17              }
18              if (processRememberme(request, response)) {
19                  // Reload page to have updated cookie in request ↩
                       to correct show remember me status
20                  user = SessionAccess.getUser(request);
21                  response.sendRedirect(request.getContextPath() + ↩
                       "/resource/wiring.jsp");
22                  return;
23              }
24              response.sendRedirect(request.getContextPath() + "/↩
```

```
                resource/wiring.jsp");
25          } else if(result.equals(LoginLogic.LoginResult.↩
               NO_NEW_LOGIN_ALREADY_LOGGED_IN)) {
26              processRememberme(request, response);
27              User user = SessionAccess.getUser(request);
28              response.sendRedirect(request.getContextPath() + "/↩
                   resource/wiring.jsp");
29          } else if(result.equals(LoginLogic.LoginResult.SUCCESS)) ↩
               {
30              processRememberme(request, response);
31              response.sendRedirect(request.getContextPath() + "/↩
                   resource/wiring.jsp");
32          } else {
33              if(request.getParameter("userName") == null){
34                  response.sendRedirect(request.getContextPath() + ↩
                       "/content/auth/login/register-or-login.jsp");
35                  return;
36              }
37              RequestDispatcher requestDispatcher = request.↩
                   getRequestDispatcher("/content/auth/login/register↩
                   -or-login.jsp?msg=" + URLEncoder.encode("Wrong ↩
                   credentials", "UTF-8"));
38              requestDispatcher.forward(request, response);
39          }
40      }
41
42      public static boolean processRememberme(HttpServletRequest ↩
           request, HttpServletResponse response) {
43          if (request.getParameter("userName") == null){
44              return true;
45          }
46          User user = SessionAccess.getUser(request);
47
48          if (request.getParameter("rememberMe") != null) {
49              LoginLogic.getInstance().setPersistentCookie(request,↩
                   response, user);
50              return true;
51          }else {
52              LoginLogic.getInstance().deletePersistentCookie(↩
                   request, response);
53              return false;
54          }
55      }
56  }
```

Quellcode 8.6: Klasse LoginServlet

```
1  public class LoginLogic {
2      public enum LoginResult{
```

```
3           SUCCESS,
4           NO_NEW_LOGIN_ALREADY_LOGGED_IN,
5           ALREADY_LOGGED_IN_WITH_TOKEN,
6           FAILED
7       }
8
9       private static LoginLogic instance = new LoginLogic();
10      public static LoginLogic getInstance() {
11          return instance;
12      }
13
14      public LoginLogic.LoginResult login(HttpServletRequest ↩
            request, HttpServletResponse response) {
15          Token token = getTokenFromCookie(request);
16          HashMap<String, String> credentials = LoginLogic.↩
                getenteredCredentials(request);
17          User alreadyLoggedinUser = SessionAccess.getUser(request)↩
                ;
18          if (token != null) {
19              User user = null;
20              try{
21                  user = DBUser.loadUser(token);
22              }catch (Exception e){
23                  e.printStackTrace();
24              }
25              if(user != null) {
26                  setUserInSession(request, response, user);
27                  //setPersistentCookie(request, response, user);
28                  return LoginResult.ALREADY_LOGGED_IN_WITH_TOKEN;
29              }
30          }
31          if (alreadyLoggedinUser == null & credentials.get("↩
                passwort") != null) {
32              String correctPassword = null;
33              try{
34                  correctPassword = DBUser.getUserpassword(↩
                        credentials.get("username"));
35                  if (correctPassword == null){
36                      return LoginResult.FAILED;
37                  }
38              }catch (Exception e){
39                  e.printStackTrace();
40              }
41              if (LoginLogic.checkIfPasswordIsCorrect(credentials.↩
                    get("passwort"), correctPassword) ) {
42                  try {
43                      setUserInSession(request, response, DBUser.↩
                            getUserByUsername(credentials.get("↩
                            username")));
```

```
44                  } catch (DBException e) {
45                      throw new RuntimeException(e);
46                  }
47                  return LoginResult.SUCCESS; // show admin page
48              }
49          }
50          if (alreadyLoggedinUser != null){
51              return LoginLogic.LoginResult.↩
                    NO_NEW_LOGIN_ALREADY_LOGGED_IN;
52          }
53          return LoginLogic.LoginResult.FAILED;
54      }
55
56      private static HashMap<String, String> getenteredCredentials(↩
            HttpServletRequest request) {
57          HashMap<String, String> credentials = new HashMap<>();
58          credentials.put("passwort", request.getParameter("p"));
59          credentials.put("username", request.getParameter("↩
                userName"));
60          return credentials;
61
62      }
63
64      public static Boolean checkIfPasswordIsCorrect(String ↩
            enteredPasswort, String correctPassword) {
65          return BCrypt.checkpw(enteredPasswort, correctPassword);
66      }
67
68      public static Token getTokenFromCookie(HttpServletRequest ↩
            request) {
69          Cookie cookie = getUserCookie(request);
70          if (cookie != null) {
71              return new Token(cookie.getValue());
72          } else return null;
73      }
74
75      public static boolean hasRememberMe(HttpServletRequest ↩
            request) {
76          Cookie cookie = getUserCookie(request);
77          if (cookie != null) {
78              return !"deleted".equals(cookie.getValue());
79          } else
80              return false;
81      }
82
83      public static Cookie getUserCookie(HttpServletRequest request↩
            ) {
84          Cookie cookies[] = request.getCookies();
85          int cookieLength = (cookies == null) ? 0 : cookies.length↩
```

```
            ;
86
87          for (int i = 0; i < cookieLength; i++) {
88              if (cookies[i].getName().equals("user")) {
89                  return cookies[i];
90              }
91          }
92          return null;
93      }
94
95      public static void setUserInSession(HttpServletRequest ↩
            request, HttpServletResponse response, User user) {
96          SessionAccess.setUser(request, user);
97      }
98
99      public static void setPersistentCookie(HttpServletRequest ↩
            request, HttpServletResponse response, User user) {
100         Cookie cookie = new Cookie("user", user.getToken().↩
                getToken() );
101         cookie.setPath("/labalive/");
102         cookie.setMaxAge(60*60*24*365); //set Age of 12 months
103         response.addCookie(cookie);
104     }
105
106     public static void deletePersistentCookie(HttpServletRequest ↩
            request, HttpServletResponse response) {
107         Cookie cookie = new Cookie("user", "deleted");
108         cookie.setPath("/labalive/");
109         cookie.setMaxAge(0); //deletes cookie
110         response.addCookie(cookie);
111     }
112 }
```

Quellcode 8.7: Klasse LoginLogic

```
1  @WebServlet(urlPatterns = { "/auth/register/" })
2  public class RegisterServlet extends BaseServlet {
3      private static final long serialVersionUID = -786677937212819↩
           0736L;
4
5      protected void dispatch(HttpServletRequest request, ↩
           HttpServletResponse response) throws ServletException, ↩
           IOException {
6          RequestContext requestContext = RequestContext.instance(↩
               request);
7          requestContext.setDirectory(Directory.AUTH);
8          try {
9              if (!checkIfUsernameIsAvailable(request)) {
10                 RequestDispatcher requestDispatcher = request.↩
```

```
                    getRequestDispatcher("/content/auth/login/register-or-login.jsp?msg=" + URLEncoder.encode("Username already in Use", "UTF-8"));
                requestDispatcher.forward(request, response);
                return;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        RegistrationLogic.RegistrationResult registerReturn = RegistrationLogic.register(request, response);
        if (registerReturn.equals(RegistrationLogic.RegistrationResult.SUCCESS)) {
            response.sendRedirect(request.getContextPath() + "/resource/wiring.jsp");
        } else {
            forward("/content/auth/login/register-or-login.jsp", request, response);

        }

    }

    public static boolean checkIfUsernameIsAvailable (HttpServletRequest request ) throws DBException {
        String username = request.getParameter("userName");
        return DBUser.getUserByUsername(username) == null;

    }

    public static boolean processRememberme(HttpServletRequest request, HttpServletResponse response) {
        User user = SessionAccess.getUser(request);

        if (request.getParameter("rememberMe") != null) {
            LoginLogic.getInstance().setPersistentCookie(request, response, user);
            return true;
        } else {
            LoginLogic.getInstance().deletePersistentCookie(request, response);
            return false;
        }
    }
}
```

Quellcode 8.8: Klasse RegisterServlet

```
public class RegistrationLogic {
```

```
2       private static Registration instance = new Registration();
3       public static Registration getInstance() {
4           return instance;
5       }
6
7       public static RegistrationLogic.RegistrationResult register(↩
            HttpServletRequest request, HttpServletResponse response) ↩
            {
8           if ("register".equals(request.getParameter("action"))) {
9               Token token = createNewToken();
10              HashMap<String, String> credentials = ↩
                    getenteredCredentials(request);
11              try {
12                  if (credentials.get("username") == null){
13                      return RegistrationResult.FAILED;
14                  }
15                  User user = DBUser.createUser(credentials.get("↩
                        username"), token, credentials.get("passwort")↩
                        );
16                  LoginLogic.setUserInSession(request, response, ↩
                        user);
17                  LoginServlet.processRememberme(request, response)↩
                        ;
18                  return RegistrationLogic.RegistrationResult.↩
                        SUCCESS;
19              } catch (TokenException | DBException e) {
20                  e.printStackTrace();
21              }
22          }
23          return RegistrationLogic.RegistrationResult.FAILED;
24      }
25
26      public enum RegistrationResult{
27          SUCCESS,
28          FAILED
29      }
30
31      private static Token createNewToken() {
32          UUID uuid = UUID.randomUUID();
33          return new Token(uuid.toString());
34      }
35
36      public static HashMap<String, String> getenteredCredentials(↩
            HttpServletRequest request) {
37          HashMap<String, String> credentials = new HashMap<>();
38          credentials.put("username", request.getParameter("↩
                userName"));
39          String password = request.getParameter("p");
40          if (password == null){
```

```
41              return credentials;
42          }
43          String salt = BCrypt.gensalt();
44          String hashedPassword = BCrypt.hashpw(password, salt);
45          credentials.put("passwort", hashedPassword);
46          return credentials;
47      }
48  }
```

Quellcode 8.9: Klasse RegistrationLogic

```
1  public class AccessCheck {
2
3      public static boolean checkAccessForUser(HttpServletRequest ↩
           request, User user){
4          long resourceID = Long.parseLong(request.getParameter("↩
               resourceID"));
5          if (resourceID <= 0){
6              return true;
7          }
8          ResourceContainer resourceContainer = ResourceProvider.↩
               getResourceById(resourceID);
9          if (user.getRolle() == User.Rollen.ADMIN){
10             return true;
11         }
12         return user != null && resourceContainer.getResource().↩
               getUserID() == user.getUserId();
13     }
14
15     public static Collection<ResourceContainer> ↩
           checkAccessForUser(Collection<ResourceContainer> ↩
           allContainer, HttpServletRequest request){
16         ResourceMap userResources = new ResourceMap();
17         User user = SessionAccess.getUser(request.getSession());
18         if (user == null){
19             Collection<ResourceContainer> vals = userResources.↩
                   values();
20             return vals;
21         }
22         if (user.getRolle() == User.Rollen.ADMIN){
23             return allContainer;
24         }
25         for(ResourceContainer resource: allContainer){
26             if(user.getUserId() == resource.getResource().↩
                   getUserID()){
27                 userResources.add(resource);
28             }
29         }
30         return userResources.values();
```

```
31      }
32
33      public static boolean verifyUser(HttpServletRequest request, ↩
            User user) throws DBException {
34          User realUser = DBUser.loadUser(user.getToken());
35          return (user.getRolle() == realUser.getRolle() && user.↩
                getUserId() == realUser.getUserId()) || (user.getRolle↩
                () == User.Rollen.ADMIN);
36      }
37  }
```

Quellcode 8.10: Klasse AccessCheck

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Literaturverzeichnis

[1] Prof. Dr.-Ing. Erwin Riederer. URL: https://www.etti.unibw.de/labalive/ (siehe Seite 1).

[2] Jetbrains. URL: https://www.jetbrains.com/idea/ (besucht am 16. 06. 2023) (siehe Seite 3).

[3] Wikipedia. URL: https://de.m.wikipedia.org/wiki/Datei:IntelliJ_IDEA_Icon.svg (besucht am 12. 06. 2023) (siehe Seite 3).

[4] Gitea. URL: https://about.gitea.com/ (besucht am 16. 06. 2023) (siehe Seite 3).

[5] Gitea. URL: https://gitea.io/images/gitea.png (besucht am 12. 06. 2023) (siehe Seite 3).

[6] Microsoft. URL: https://code.visualstudio.com/ (besucht am 16. 06. 2023) (siehe Seite 4).

[7] Visualstudio. URL: https://code.visualstudio.com/assets/images/code-stable.png (besucht am 12. 06. 2023) (siehe Seite 4).

[8] Microsoft. URL: https://www.microsoft.com/en-us/microsoft-365/visio/visio-in-microsoft-365 (besucht am 16. 06. 2023) (siehe Seite 4).

[9] Wikimedia. URL: https://upload.wikimedia.org/wikipedia/commons/9/9e/Microsoft_Visio_Logo_%282013-2019%29.png (besucht am 12. 06. 2023) (siehe Seite 4).

[10] Microsoft. URL: https://www.microsoft.com/en-us/microsoft-365/access (besucht am 16. 06. 2023) (siehe Seite 4).

[11] Wikipedia. URL: https://de.wikipedia.org/wiki/Microsoft_Access#/media/Datei:Microsoft_Office_Access_(2019-present).svg (besucht am 12. 06. 2023) (siehe Seite 4).

[12] Oracle. URL: https://www.oracle.com/java/ (besucht am 16. 06. 2023) (siehe Seite 5).

[13] Hirschtec. URL: https://hirschtec.eu/wp-content/uploads/2018/05/hirschtec.eu-00_agile-umsetzung-java-logo-icon.png#iLightbox[postimages]/0 (besucht am 12. 06. 2023) (siehe Seite 5).

[14] MDN Web Docs.
URL: https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML (besucht am 16. 06. 2023) (siehe Seite 5).

[15] MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript (besucht am 16. 06. 2023) (siehe Seite 5).

[16] W3schools. URL: https://www.w3schools.com/sql/sql_intro.asp (besucht am 16. 06. 2023) (siehe Seite 5).

[17] Oracle. URL: https://docs.oracle.com/javaee/7/tutorial/servlets001.htm#BNAFE (besucht am 22. 06. 2023) (siehe Seite 22).

[18] Oracle. URL: https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServlet.html (besucht am 24. 06. 2023) (siehe Seiten 23, 25).

[19] MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Session (besucht am 20. 06. 2023) (siehe Seite 26).

[20] OWASP. URL:
https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (besucht am 20. 06. 2023) (siehe Seite 29).

[21] Coda Hale.
URL: https://codahale.com/how-to-safely-store-a-password/ (besucht am 20. 06. 2023) (siehe Seite 29).

[22] OWASP. URL: https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html (besucht am 22. 06. 2023) (siehe Seite 30).

[23] MDN Web Docs.
URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/password (besucht am 22. 06. 2023) (siehe Seiten 30, 31).

Stichwortverzeichnis