# IPA-Simulation

Raphael Pfeffinger

## Inhaltsverzeichnis

Inhalt

[IPA-Simulation 1](#_Toc169858855)

[Inhaltsverzeichnis 2](#_Toc169858856)

[1. Projektmanagement 3](#_Toc169858857)

[1.1 Projektmethodik 3](#_Toc169858858)

[1.2 Zeitplanung 1](#_Toc169858859)

[1.3 Testkonzept 1](#_Toc169858860)

[1.3.1 Testziele 1](#_Toc169858861)

[1.3.2 Teststufen 1](#_Toc169858862)

[1.3.3 Testobjekte 1](#_Toc169858863)

[1.3.4 Testfälle 2](#_Toc169858864)

[1.4 Arbeitsjournal 3](#_Toc169858865)

[2. Projektdokumentation 3](#_Toc169858866)

[2.1 Informieren (Analyse) 3](#_Toc169858867)

[2.2 Planen (Design) 5](#_Toc169858868)

[2.2.1 Mockups 7](#_Toc169858869)

[2.3 Entscheiden 9](#_Toc169858870)

[2.3.1 Backup Lösung 9](#_Toc169858871)

[2.3.2 Login Security 11](#_Toc169858872)

[2.4 Realisieren (Implementieren) 12](#_Toc169858873)

[2.4.1 Datenbank erstellen 12](#_Toc169858874)

[2.4.2 Startseite, Login Seite, Registrierseite 13](#_Toc169858875)

[2.4.3 API Methoden Zentral im Frontend 15](#_Toc169858876)

[2.4.4 Hash Algorithmus 16](#_Toc169858877)

[2.4.5 Session Konfiguration 17](#_Toc169858878)

[2.5 Kontrollieren (Test) 18](#_Toc169858879)

[2.6 Auswerten 19](#_Toc169858880)

[3. Abbildungsverzeichnis 20](#_Toc169858881)

# Projektmanagement

## Projektmethodik

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte BeschreibungAls Projektmethode verwende ich das IPERKA Modell. Die klare Struktur dieses Modells hilft bei der Umsetzung des Projekts im Ganzen. Die Informationsbeschaffung wird im 2. Teil des Dokuments klarer beschrieben werden, und ist ebenfalls im Arbeitsjournal ersichtlich. Die Planung im Sinne der Zeit befindet sich im ersten Teil und wird fortgeführt im 2. Teil mit der Planung der Komponenten und Anwendungsstrategien jener Komponenten. Die Restlichen Punkte sind ausschliesslich in der Dokumentation enthalten. IPERKA wird grundsätzlich als Projektmethodik verwendet und auch als Gerüst der Dokumentation. Im Speziellen verwende ich für die Entwicklung das Wasserfallmodell, welches IPERKA sehr ähnelt.

Abb. 1: IPERKA Grafik

Abb. 1 IPERKA Grafik 1

Abbildung 1 IPERKA Grafik

## Zeitplanung

Zeitplanung in Form eines Balkendiagramms mit den Schritten des Wasserfallmodells.

Die Tage sind in 4 Abschnitte unterteilt, 2 am Morgen (8:30-10:35 & 10:50-12:15) und 2 am Nachmittag (13:15-15:00 & 15:10-17:30). Im Diagramm sind die einzelnen Schritte von 1-4 bewertet und entsprechen den tagesabschnitten. Die Dokumentation ist immer Inbegriffen, sowie das Arbeitsjournal.



## Testkonzept

### Testziele

|  |  |  |  |
| --- | --- | --- | --- |
| Nr. | Beschreibung | Messgrösse | Priorität\* |
| 1 | Schnittstellen test erfolgreich | Korrektheit der Daten in der Datenbank oder eingegeben | 1 |
| 2 | Login test erfolgreich | Session aktiviert, andere Startseite | 1 |
| 3 | Testdaten erfolgreich | Falsche daten lassen sich nicht einfügen | 2 |
| 4 | Registrierung erfolgreich | Daten korrekt mit DB ausgetauscht | 1 |

Priorität: 1=hoch, 2=mittel, 3=tief

### Teststufen

Verwendet werden hauptsächlich Integrationstests und wo möglich Unittests. Zuerst werden Unittests in einer Testumgebung durchgeführt, wo möglich. Als nächstes mache Ich Integrationstests mit dem Ziel die korrekte Zusammenarbeit zwischen DB und API zu garantieren.

### Testobjekte

|  |  |  |
| --- | --- | --- |
| Nr. | Objekt | Beschreibung |
| 1 | Login und Registrierungsseite | Die Eingabefelder mit den Restriktionen müssen getestet werden |
| 2 | API | Die REST API muss getestet werden ob alle CRUD-Funktionen die Daten korrekt verarbeiten. |
| 3 | Datenbank | Die Datenbank muss mit Testdaten gefüttert werden |
| 4 | Sicherheitslayer | Der Sicherheitslayer mit dem Erstellen des Cookies und der Session muss getestet werden |

### Testfälle

|  |  |  |  |
| --- | --- | --- | --- |
| ID/Bezeichnung | 001 | Öffnen der Login Seite |  |
| Beschreibung | Die Login Seite muss geöffnet werden und die Felder sollten verfügbar sein. | | |
| Testvoraussetzung | Programm muss gestartet sein, die Startseite muss geöffnet sein | | |
| Testschritte | 1. In die URL-Leiste /Login eingeben (oder auf Login Link klicken) | | |
| Erwartetes Ergebnis | Die Login Seite öffnet sich und der Button ist deaktiviert, da Felder leer. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID/Bezeichnung | 002 | Öffnen der Register Seite |  |
| Beschreibung | Die Registrierungsseite muss garantieren das der Besucher sich einen Account einrichten kann. | | |
| Testvoraussetzung | Programm muss gestartet sein und Startseite geöffnet | | |
| Testschritte | 1. In die URL-Leiste /Registrieren eingeben (oder auf link auf der Startseite klicken) | | |
| Erwartetes Ergebnis | Die Registrierungsseite öffnet sich und die Felder sind deaktiviert und leer. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID/Bezeichnung | 003 | Erstellen von Account und Login |  |
| Beschreibung | Der Benutzer soll einen Account erstellen, um sich danach einloggen zu können | | |
| Testvoraussetzung | Registrierungsseite muss geöffnet sein und Felder sind deaktiviert (Testfall: 002). | | |
| Testschritte | 1. Valide Daten in die Felder Eingeben 2. Auf Button Speichern klicken 3. Zu Login Seite gehen 4. Mit den gespeicherten Daten Anmelden | | |
| Erwartetes Ergebnis | Wenn die Daten korrekt eingegeben werden und speichern gedrückt wird soll zum Start zurückgebracht werden.  Nach Login soll wieder auf Startseite gebracht werden, wo ein grosser Text mit «Hello» stehen muss. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID/Bezeichnung | 004 | Negativtest von Anmeldedaten |  |
| Beschreibung | Um zu gewährleisten, dass die Restriktionen der Eingebbaren Daten funktionieren, führe ich einen Negativtest durch. | | |
| Testvoraussetzung | Registrierungsseite geöffnet, Felder deaktiviert. | | |
| Testschritte | 1. Als E-Mail keine E-Mail angeben 2. Als Passwort etwas einfaches wie «Passwort» eingeben | | |
| Erwartetes Ergebnis | Das Formular sollte nicht speicherbar sein resp. Es kommt eine Fehlermeldung mit der Meldung, dass die Daten nicht den Voraussetzungen entsprechen. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID/Bezeichnung | 005 | Testen von API CRUD |  |
| Beschreibung | Die API CRUD Funktionen müssen getestet werden. Dafür gibt es einen automatischen Test | | |
| Testvoraussetzung | 1. Postman muss installiert und geöffnet sein 2. Die Postman Collection und Environment müssen in Postman geöffnet sein. | | |
| Testschritte | 1. Nach öffnen der Collection Rechtsklick auf Ordner 2. Dann auf run | | |
| Erwartetes Ergebnis | Es Sollte alle API-Funktionen testen, welche sich in der Collection befinden und überall sollte das Ergebnis 200 OK sein. | | |

## Arbeitsjournal

### Montag, 17.06.2024

|  |  |  |  |
| --- | --- | --- | --- |
| Uhrzeit | Soll | Ist | Probleme |
| 8:30-10:30 | Studieren des Leitfadens und Kriterienkatalogs der IPA, sowie erstellen des Dokuments mit Inhaltsverzeichnis. | Studieren des Leitfadens und Kriterienkatalogs der IPA, sowie erstellen des Dokuments mit Inhaltsverzeichnis. | keine |
| 10:50-12:15 | Erstellen von Zeitplan und Erstellen von der Informationsbeschaffung in der Dokumentation | Erstellen der Informationen und Kurzbeschrieb des Produkts | Zeitplan noch nicht erstellt da fehlende Kenntnis über Diagramm |
| 13:15-15:00 | Planung und Entscheidung der Komponenten, Erstellen von Datenbank Diagramm | Planung und Entscheidung der Komponenten, Erstellen von Datenbank Diagramm | keine |
| 15:00-17:30 | Erstellen der Projektmappe und des Repository und der Datenbank container | Erstellen der Projektmappe und des Repository und der Datenbank container | keine |

Dem Leitfaden entsprechend habe ich ein Dokument erstellt und begonnen die benötigten Punkte abzuhaken. Da ich noch keine genaue Ahnung von der Struktur hatte habe ich ein bisschen kreuz und quer die Punkte abgehakt. Ich habe mit der Dokumentation begonnen und die nötigen Informationen über mögliche Sicherheitsrisiken bei einer Login Seite gesammelt.

Probleme gab es erst bei der Erstellung des Backups über GitHub. Aus Irgendeinem Grund, der mir nicht ganz klar ist, überschneiden sich meine 2 Accounts. Bei einem Push in das Repository bekam ich permission denied für meinen Schulaccount, obwohl ich im Config file von Git mit dem Privataccount angemeldet war. ich habe das Problem gelöst in dem ich den Schulaccount als Collaborator eingeladen habe.

Bei der Erstellung der Projektmappe in Visual Studio hatte ich keine Probleme.

**Noch zu machen:** da ich einen Falschen Ansatz beim Zeitplan hatte muss ich das Morgen, Dienstag, dem 18.06.2024, nochmals überarbeiten.

**Schüler:** Raphael Pfeffinger

|  |  |  |  |
| --- | --- | --- | --- |
| Uhrzeit | Soll | Ist | Probleme |
| 8:30-10:30 | Überarbeiten von Zeitplan und des Arbeitsjournals | Überarbeiten von Zeitplan und des Arbeitsjournals | keine |
| 10:50-12:15 | Erstellen des Testkonzepts  Umstrukturierung der Dokumentation | Erstellen des Testkonzepts  Umstrukturierung der Dokumentation | keine |
| 13:15-15:00 | Punkt der Entscheidung Abschliessen  Beginnen mit der Arbeit am Produkt | Punkt Entscheidung abgeschlossen  Datenbank wurde erstellt und Verbindung aufgebaut | Der datenbankcontainer wurde anfangs nicht auf dem Port 1433 exposed. Die Lösung war den Port festzulegen in der docker-compose.override.yaml datei |
| 15:00-17:30 | Erstellen der Context datei  Configurieren von connection string  Erstellen der Stored procedures | Erstellen der Context datei  Configurieren von connection string | Aufgrund der Verbindungsprobleme mit der DB, muss ich die Stored Procedures nach hinten verschieben. |

### Dienstag, 18.06.2024

Das Dokument für die Dokumentation und die Planung des Projekts, habe ich überarbeitet. Der Zeitplan wurde erneuert und angepasst. Ausserdem habe ich das Testkonzept erstellt nach HERMES Standard. Die Verschiedenen Test-Cases werden laufend angewandt, um die Sicherheit der Applikation sicherzustellen.

Die oben erwähnten Verbindungsprobleme konnten behoben werden. Die Genaue Lösung ist in der Dokumentation ersichtlich. Im Prinzip ging es darum, dass der Datenbank-Container nicht auf dem Port 1433 exposed wurde.

**Noch zu machen:** Die Stored Procedures müssen noch erstellt werden und die Testdaten für die Datenbank Tabelle.

### Mittwoch, 19.06.2024

|  |  |  |  |
| --- | --- | --- | --- |
| Uhrzeit | Soll | Ist | Probleme |
| 8:30-10:30 | Erstellen der Formulare | Erstellen der Formulare | keine |
| 10:50-12:15 | Registriermechanismus integrieren | Registriermechanismus integrieren | keine |
| 13:15-15:00 | Login mit Hash versehen und integrieren | Hashfunktion erstellt aber abgleich und Session konfiguration noch nicht | keine |
| 15:00-17:30 | Sicherheitstoken und Session erstellen | Session wurde konfiguriert ist aber nicht fertig. Token wird nicht mehr realisiert | Unterschätzt wie lange es ging |

Heute war zuerst Aufräumen dran, bevor ich weiter machen konnte. Ich musste den Zeitplan aktualisieren, weil meine vorherige Ausgabe sehr begrenzt war.

Unter anderem konnte ich die Seiten erzeugen für die Formulare und konnte die Helferfunktionen, welche ich benutzen werde für die API, erstellen.

**Noch zu machen:** die Formulare für Login und Registrierung, das Überprüfen der Formulare, der Login Mechanismus mit hash und salt, das Erstellen einer Session.

**Schüler:** Raphael Pfeffinger

### Donnerstag, 20.06.2024

|  |  |  |  |
| --- | --- | --- | --- |
| Uhrzeit | Soll | Ist | Probleme |
| 8:30-10:30 | Erstellen der Formulare | Erstellen der Formulare | keine |
| 10:50-12:15 | Registriermechanismus integrieren | Registriermechanismus integrieren | keine |
| 13:15-15:00 | Login mit Hash versehen und integrieren | Hashfunktion erstellt aber abgleich und Session konfiguration noch nicht | keine |
| 15:00-17:30 | Sicherheitstoken und Session erstellen | Session wurde konfiguriert ist aber nicht fertig. Token wird nicht mehr realisiert | Unterschätzt wie lange es ging |

Bei dem Erstellen der Formulare gab es keine Probleme. Ich habe mich mit der begrenzten Zeit so gut es ging an die Mockups gehalten.

Der Login Mechanismus stellte schon eine grössere Herausforderung dar. Da die Passwörter zuerst mit einem zufälligen Salt verschlüsselt wurden, konnte ich sie beim Login nicht vergleichen. Darum entschied ich mich die User ID als Salt zu verwenden. So habe ich nicht immer den gleichen Salt, aber für den ausgewählten User ist es der gleiche und damit der gleiche Hash.

Weiterhin fiel die Session Konfiguration aufgrund des Zeitdrucks schwerer. Ich habe es so gut es ging hinbekommen, jedoch wurde es nicht fertig.

**Noch zu machen:** Dokumentation, Abbildungsverzeichnis

**Schüler:** Raphael Pfeffinger

### Freitag, 21.06.2024

|  |  |  |  |
| --- | --- | --- | --- |
| Uhrzeit | Soll | Ist | Probleme |
| 8:30-10:30 | Nachführen Dokumentation  Abbildungsverzeichnis einfügen | Nachführen Dokumentation  Abbildungsverzeichnis einfügen | keine |
| 10:50-12:15 | Feinschliff Dokumentation  Einfügen Zeitplan  Einfügen Arbeitsjournal | Feinschliff Dokumentation  Einfügen Zeitplan  Einfügen Arbeitsjournal | keine |

# Projektdokumentation

## Informieren (Analyse)

Ziel des Projekts ist das Bauen eines Logins. Das Login ist nicht einfach normal, sondern es ist auf Sicherheit bedacht. Das bedeutet, nach dem Bauen der Grundstruktur mit Start-, Login- und Registrierungsseite, werden die grössten Sicherheitsprobleme, welche mit dem Login zusammenhängen ausgemerzt. Ziel ist es zu verstehen, was die grössten Lücken heutzutage sind.

Zu den häufigsten Risiken gehören laut Strongdm.com [[1]](#footnote-1)die folgenden:

* **Brute-Force Attack**: Eine Attacke bei der versucht wird das Passwort mit einer Wörterliste herauszufinden, indem man Anfrage um Anfrage schickt.

**Lösung:** die Login Versuche limitieren, 2-Faktor-Atuhentifizierung und starke Passwörter

* **Schwache Credentials**

**Lösung:** beim Login Passwortstärke vorgeben (z.B. Menge an Zeichen, Sonderzeichen, Zahlen usw.)

* **Poor Session Management:** Session identifiers sind nicht geschützt, Sessions können von Angreifern übernommen werden, Sessions werden nicht zerstör bzw. sind immer valid.

**Lösung:** eindeutige Session identifiers, sicherstellen dass Sessions zerstört und gelöscht werden.

* **Staying logged in & Cookie hijacking:** wenn es einen remember me button gibt, können Angreifer das Erstellungsmuster von Cookies erraten und den Account übernehmen mithilfe von XSS (Cross site scripting) und Brute force Attacks.

**Lösung:** cookie verschlüsseln, den Remember me button weglassen.

* **SQL-Injections:** Methode bei der Mithilfe von SQL-Befehlen im Login bestimmte Accounts übernommen werden.

**Lösung:** Parameter basierte Datenbank, Stored Procedures

## Planen (Design)

In der Zeit halte ich mich an den Zeitplan aus dem 1. Teil des Dokuments. Eine Rollenzuweisung gibt es nicht, da ich allein arbeite.

**Datenbankmodell (ERM):**

**Ein Bild, das Text, Schrift, Screenshot, Zahl enthält.

Automatisch generierte Beschreibung**

Abbildung 2 Datenbank Tabelle

**UML Frontend Klassendiagramm:**

**Ein Bild, das Text, Diagramm, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung**

Abbildung 3 UML Klassendiagramm Frontend

**Ein Bild, das Text, Screenshot, Schrift, Diagramm enthält.

Automatisch generierte BeschreibungUML API Klassendiagramm:**

Abbildung 4 UML Klassendiagramm API

### Mockups

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Abbildung 5 Mockup Login & Register

Ein Bild, das Text, Screenshot, Schrift, Design enthält.

Automatisch generierte Beschreibung

Abbildung 6 Mockup Passwort vergessen

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Abbildung 7 Mockup neues Passwort

## Entscheiden

Die Applikation wird in ASP.NET Core umgesetzt und in C# geschrieben. Das liegt daran, dass dieses Framework mir bereits bekannt ist, und mit C# kann ich gut Arbeiten, ausserdem geht das Ganze nach MVC-Architektur. Die API ist eine REST-Schnittstelle und hat einen OData Standard. Die Datenbank ist in MSSQL geschrieben und läuft auf SSMS (SQL Server Management Studio).

Server, Datenbank und API laufen jeweils in einem Docker Container.

Für die Versionierung und Sicherung verwende ich GitHub[[2]](#footnote-2).

### Backup Lösung

Wie oben erwähnt wird GitHub verwendet für die Versionierung. Es dient auch zur Sicherheit als Backup. Die Dokumente für die Projektdokumentation befinden sich ebenfalls im Repository und können jederzeit wiederhergestellt werden.

**Beweis:**

**Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung**

Abbildung 8 Github Repository Beweis

**Dokumentation:**

**Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung**

Abbildung 9 Github repository Dokumentation

**Projekt:**

**Ein Bild, das Text, Screenshot, Software, Zahl enthält.

Automatisch generierte Beschreibung**

Abbildung 10 Github Repository Source

### Login Security

Als Security Lösungen für das Login dient die bei kap. 2.3 genannte Liste als Orientierung. Insgesamt werden im ersten Schritt folgende implementiert:

* Stored Procedures
* Password and E-Mail Restrictions (mind. 10 Zeichen mit Zahl und Sonderzeichen)
* Session Management mit automatischer Zerstörung der Session
* Verschlüsseltes Cookie

## Realisieren (Implementieren)

### Datenbank erstellen

Ich habe als ersten Schritt die Datenbank aufgesetzt in einem Container. Es handelt sich um MSSQL. Danach habe ich die Datenbank Datei erstellt und die erste und einzige Tabelle eingefügt.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Abbildung 11 Datenbank Code

**Probleme:**

Der Datenbank Container wollte nicht exposed werden auf dem Port 1433. Die Lösung dazu war im docker-compose.override.yaml den Port zu definieren.

Ein Bild, das Text, Schrift, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

Abbildung 12 docker.compose.override.yaml Datei

Diese Datei wird von Visual Studio automatisch erzeugt, wenn eine Applikation Containerized wird. Sie wird erstell zu debugg zwecken und wird ebenfalls von Docker-Compose gelesen.

### Startseite, Login Seite, Registrierseite

Zuerst habe ich die Login Seite erstellt. Dabei gab es im Prinzip keine Probleme. Ich habe das Standard-Layout der shared View von Asp.NET Core verwendet und das Login Formular so gut es ging in der begrenzten Zeit nach den Mockups gestaltet.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Abbildung 13 Login Seite UI

**Probleme:** Wegen Zeitmangel konnte ich nicht noch die 2-Faktor-Authentifizierung einbauen. Deswegen fällt dieses Mockup weg.

Die nächsten Schritte war die Registrierungsseite. Diese wird ähnlich aufgebaut wie die Login Seite.

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 14 Registrierseite UI

**Ablauf:** mit den Testdaten: [max.mustermann@gmail.com](mailto:max.mustermann@gmail.com), Passwort, Raphael, Pfeffinger, wird ein neues Konto erzeugt. Dabei wird das Passwort gehashed mit einer Hash Funktion (kap. 2.4.4.). Danach geht der Datensatz per POST-Funktion an die API und darüber in die Datenbank.

Ist der Benutzer gespeichert, kann man sich einloggen. Beim Login werden in der Authenticate() Funktion, der Datensatz aus der Datenbank genommen, welcher die eingegebene E-Mail hat. Dann wird das eingegebene Passwort gehashed. Wichtig dabei ist, dass der salt der gleiche sein muss wie beim Hash-Vorgang beim Speichern des Passworts. Denn damit stimmen die Hashes überein. Nach Vergleich wird eine Session gestartet und darin die E-Mail des eingeloggten gespeichert.

Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung

Abbildung 15 Code Authenticate Funk. in LoginController

### API Methoden Zentral im Frontend

Um die Methoden auf der API zu Autorisieren, wollte ich im seclogin (Web-Projekt) die Methoden, welche die API direkt ansprechen zentral machen. Also im Prinzip als Helper Klasse. Da die Token Autorisieren wegfällt, mache ich das trotzdem.

Ich habe ein Interface erstellt, welches im Program.cs als HttpClient konfiguriert wird. Die dazugehörige ApiHelper.cs Klasse erweitert diese Methoden und initialisiert sie.

Die Methoden benötigen auf jeden Fall einen Parameter URL und bei POST und PATCH einen Parameter data in Json Format.

**Beispiel:**

Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung

Abbildung 16 Zentrale API Funk. Bsp.

### Hash Algorithmus

Ein Bild, das Text, Screenshot, Software, Betriebssystem enthält.

Automatisch generierte Beschreibung

Abbildung 17 Hash Algorithmus

Wie auf dem Bild zu sehen ist, verlangt die Hash-Funktion einen Parameter Salt. Das Salt soll verhindern, dass Hashes bei einem Angriff verglichen werden können. Deshalb wird eine für den Angreifer unbekannte Zeichenkette dem Passwort angehängt und dann gehashed. In Meinem Fall ist das Salt die ID des Benutzers. Der Nachteil Dabei ist, dass der PRIMARY KEY im Registrierungscontroller erzeugt werden muss, um zu garantieren, dass der Hash der gleiche sein kann. Weitere Varianten waren, dass das Passwort in der Datenbank gehashed wird, d.h. es würde Klartext an die Datenbank übertragen werden, was eine potenzielle Sicherheitslücke darstellt. Weiterhin um den Salt noch sicherer zu machen, könnte man den Salt mit dem Passwort speichern und später beim Login beide holen. Jedoch wäre der Salt dann ebenfalls im Klartext.

Ich habe mich für die erste Version entschieden, weil es für mich die Sicherste Variante darstellt.

### Session Konfiguration

Nach dem Login soll eine Session gestartet werden. Dafür muss sie zuerst konfiguriert werden. Eigentlich soll noch ein Logout Button kommen, jedoch fehlte mir dafür sehr knapp die Zeit.

**Session Konfiguration: Ein Bild, das Text, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung**

Abbildung 18 Session Konfiguration

Wie zu sehen ist, ist die Session 5 Stunden gültig. Zusätzlich zu Session sollten noch eine Cookie-Konfiguration und ein Token generiert werden. Mit dem Token könnte man dann die API-Funktionen Autorisieren, um unbefugte Benutzung zu verhindern.

## Kontrollieren (Test)

**Testcase: 001**

**Erwartetes Resultat:** geöffnete Login Seite mit Formular

**Tatsächliches Resultat:** geöffnete Login Seite mit Formular

**Testcase: 002**

**Erwartetes Ergebnis:** geöffnete Registrierseite mit Formular.

**Tatsächliches Ergebnis:** geöffnete Registrierseite mit Formular

**Testcase: 003**

**Erwartetes Ergebnis:** Nach Login zum Start zurückgebracht werden

**Tatsächliches Ergebnis:** wird nicht zu Start zurückgebracht

**Testcase: 004**

**Erwartetes Ergebnis:** Daten können nicht gespeichert werden.

**Tatsächliches Ergebnis:** Daten können nicht gespeichert werden.

**Testcase: 005**

**Erwartetes Ergebnis:** alle Requests geben OK 200

**Tatsächliches Ergebnis:** alle Requests geben OK 200

Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung

Abbildung 19 Testbeweis API

## Auswerten

Ich konnte das Projekt leider nicht ganz so umsetzen wie ich anfangs wollte. Das Problem war nicht die mangelnde Kenntnis, sondern das Zeitmanagement. Ich habe unterschätzt wie viel Zeit die Entwicklung alleine in Anspruch nimmt. Rechnen wir dann noch die Testphase dazu, welche genauso lang wie die Entwicklung sein sollte, wird es schwierig die geplanten Ziele umzusetzen.

Konkret konnte ich bestimmte Sicherheitsmechanismen, welche den Kern der Arbeit darstellen sollten, nicht einbauen. Die 2-Faktor-Authentifizierung wäre eine gute Chance gewesen, um noch besser zu verstehen, wie der Login- und Registrierungsprozess funktioniert.

Ich muss allerdings auch sagen, dass die Grundfunktion besteht und dass die verschiedenen Mängel, welche ich nicht aufgezählt habe, eigentlich nur Feinschliff sind.

Ich nehme aus dem Projekt mit, dass Zeitmanagement unabdingbar ist und direkt am Anfang eines Projekts ein Zeitplan zu erstellen ein Muss ist.

# Abbildungsverzeichnis

[Abbildung 1 IPERKA Grafik 3](#_Toc169857856)

[Abbildung 2 Datenbank Tabelle 9](#_Toc169857857)

[Abbildung 3 UML Klassendiagramm Frontend 9](#_Toc169857858)

[Abbildung 4 UML Klassendiagramm API 10](#_Toc169857859)

[Abbildung 5 Mockup Login & Register 11](#_Toc169857860)

[Abbildung 6 Mockup Passwort vergessen 11](#_Toc169857861)

[Abbildung 7 Mockup neues Passwort 12](#_Toc169857862)

[Abbildung 8 Github Repository Beweis 13](#_Toc169857863)

[Abbildung 9 Github repository Dokumentation 14](#_Toc169857864)

[Abbildung 10 Github Repository Source 14](#_Toc169857865)

[Abbildung 11 Datenbank Code 15](#_Toc169857866)

[Abbildung 12 docker.compose.override.yaml Datei 15](#_Toc169857867)

[Abbildung 13 Login Seite UI 16](#_Toc169857868)

[Abbildung 14 Registrierseite UI 17](#_Toc169857869)

[Abbildung 15 Code Authenticate Funk. in LoginController 17](#_Toc169857870)

[Abbildung 16 Zentrale API Funk. Bsp. 19](#_Toc169857871)

[Abbildung 17 Hash Algorithmus 20](#_Toc169857872)

[Abbildung 18 Session Konfiguration 21](#_Toc169857873)

[Abbildung 19 Testbeweis API 22](#_Toc169857874)

1. <https://www.strongdm.com/blog/authentication-vulnerabilities> [↑](#footnote-ref-1)
2. https://github.com/ruffy777/Secure\_Login.git [↑](#footnote-ref-2)