Probe IPA

M223

Siemens

Jordan Dainese

Inhaltsverzeichnis

[3 Vorwort 3](#_Toc130548650)

[4 Aufbau und Ablauf (Teil 1) 4](#_Toc130548651)

[4.1 Ausgangslage 4](#_Toc130548652)

[4.2 Aufgabenstellung 4](#_Toc130548653)

[4.2.1 Anforderungen 4](#_Toc130548654)

[4.2.2 Akzeptanzkriterien 4](#_Toc130548655)

[4.2.3 Funktionale Anforderungen 4](#_Toc130548656)

[4.2.4 Nicht-Funktionale Anforderungen 5](#_Toc130548657)

[4.2.5 Individuelle Bewertungskriterien 5](#_Toc130548658)

[4.2.6 Technologien 7](#_Toc130548659)

[4.2.7 Versionierung 7](#_Toc130548660)

[4.3 Projektaufbauorganisation 7](#_Toc130548661)

[4.3.1 Projektmitglieder 7](#_Toc130548662)

[4.3.2 Projektmethode 7](#_Toc130548663)

[4.4 Zeitplan 8](#_Toc130548664)

[4.4.1 Vorgegebene Termine 8](#_Toc130548665)

[4.4.2 Zeitplan 9](#_Toc130548666)

[4.5 Arbeitsjournal 10](#_Toc130548667)

[5 Projekt (Teil 2) 16](#_Toc130548668)

[5.1 Kurzzusammenfassung 16](#_Toc130548669)

[5.1.1 Ausgangslage 16](#_Toc130548670)

[5.1.2 Umsetzung 16](#_Toc130548671)

[5.2 Informieren 16](#_Toc130548672)

[5.2.1 Ziele der Aufgabenstellung 16](#_Toc130548673)

[5.3 Planen 17](#_Toc130548674)

[5.3.1 Sitemap 17](#_Toc130548675)

[5.3.2 Testkonzept 17](#_Toc130548676)

[5.4 Entscheiden 21](#_Toc130548677)

[5.4.1 Varianten 21](#_Toc130548678)

[5.4.2 Risiken 21](#_Toc130548679)

[5.4.3 Entscheid 22](#_Toc130548680)

[5.5 Realisieren 22](#_Toc130548681)

[5.5.1 Klassendiagramm 22](#_Toc130548682)

[5.5.2 Umsetzungen 22](#_Toc130548683)

[5.5.3 Nicht umgesetzt 24](#_Toc130548684)

[5.6 Kontrollieren 24](#_Toc130548685)

[5.6.1 Manuelle Tests 24](#_Toc130548686)

[5.7 Auswerten 24](#_Toc130548687)

[5.8 Reflexion 25](#_Toc130548688)

[5.8.1 Erfolge 25](#_Toc130548689)

[5.8.2 Misserfolge 25](#_Toc130548690)

[5.8.3 Verbesserungsmöglichkeiten 25](#_Toc130548691)

[5.8.4 Fazit 25](#_Toc130548692)

[6 Quellenverzeichnis 25](#_Toc130548693)

[7 Glossar 25](#_Toc130548694)

[8 Abbildungsverzeichnis 26](#_Toc130548695)

# Vorwort

Diese Dokumentation gehört zum Modul-223 und wurde als Probe IPA von Jordan Dainese umgesetzt. In dieser Dokumentation ist das Vorgehen und der Ablauf der Arbeit an der Agenda Applikation beschrieben.

Die Dokumentation ist in 2 Teile gegliedert.

Im ersten Teil werden die ausgearbeitete Aufgabenstellung und der Ablauf der Arbeit aufgezeigt. Ebenfalls ist ersichtlich, mit welchen Mitteln gearbeitet wurde und welche Vorkenntnisse bereits vorhanden sind.

Im zweiten Teil ist die Dokumentation der eigentlichen Arbeit niedergeschrieben, sowie die aufgetretenen Hindernisse und das Testing der Applikation.

Die Arbeit wurde von Anfang bis Schluss mit der Projektplanungsmethode IPERKA geplant.

Die Auswertungsphase wird mit einer Reflexion und Fazit abgeschlossen, welche am Schluss stehen.

# Aufbau und Ablauf (Teil 1)

## Ausgangslage

Im Modul 223 geht es darum eine IPA zu simulieren, damit wir Lernenden Erfahrungen sammeln könne, die uns helfen uns auf die richtige IPA vorzubereiten. Da das Zeitfenster jedoch etwas kürzer ist, sind auch die Anforderungen etwas weniger. Dabei kann man ein Softwareprojekt nach eigenen Wünschen gestalten. Diese soll dieselben Kriterien, wie aus dem Standardkriterienkatalog erfüllen und noch zusätzlich 3 aus dem individuellen Kriterienkatalog, die man selbst bestimmen darf. Ich habe mich dazu entschieden eine Agendatool als Weblösung zu erstellen. Dabei kann man einen Account erstellen und eigene Termine eintragen und bearbeiten.

## Aufgabenstellung

### Anforderungen

* Front- und Backend
* Zentrale Datenbank
* Mehrere User greifen gleichzeitig auf den gleichen Datenbestand zu
* Zentrale Benutzer- und Rechteverwaltung

### Akzeptanzkriterien

* Realisierung der Applikation
* Saubere Dokumentation

### Funktionale Anforderungen

* Ein Benutzer kann sich anmelden
* Ein Benutzer kann sich registrieren
* Ein Benutzer kann eigene Termine erstellen
* Ein Benutzer kann eigene Termine editieren
* Ein Benutzer kann eigene Termine löschen
* Ein Benutzer hat eine Übersicht über all seine Termine
* Erstellte Termine werden in einer Datenbank gespeichert
* Beim wieder Einloggen eines Benutzers werden seine Termine aus der Datenbank geladen
* Ein Adminbenutzer hat Zugriff auf ein Adminpanel
* Mit dem Adminpanel kann ein Admin Nutzer bearbeiten (erstellen, löschen, editieren)
* Ein Admin kann weitere Adminbenutzer erstellen

### Nicht-Funktionale Anforderungen

* Die Applikation soll leicht zu bedienen sein
* Der Code ist strukturiert und kommentiert

### Individuelle Bewertungskriterien

Ich habe mich für folgende individuelle Kriterien aus dem Katalog entschieden:

1. Individuelles Bewertungskriterium

|  |  |
| --- | --- |
| *Nummer Katalog-Kriterium - Bezeichnung* | |
| 166 - Codingstyle - lesbarer Code | |
| *Definition (Leitfrage)* | |
| Ist der Code lesbar geschrieben, gut gegliedert und ist die Namensgebung gut gewählt? | |
| *Gütestufe 3* | *Gütestufe 2* |
| Die Namensgebung entspricht den Vorgaben oder ist einfach gut gewählt. Die Struktur des Codes ist ebenfalls gemäss möglicher Richtlinien oder einfach übersichtlich gemacht. Es ist eine gewisse Einheit zu sehen in der Art und Weise, wie der Code strukturiert ist (d.h. es ist überall etwa gleich gemacht). | Die Namensgebung ist ab und zu ungeschickt gewählt, Vorgaben sind teilweise berücksichtigt. Die Codestruktur ist uneinheitlich (so dass Lesbarkeit leidet). |
| *Gütestufe 1* | *Gütestufe 0* |
| Die Namensgebung ist öfters verwirrend oder unpräzise. Dem Code fehlt es an einigen Stellen an klarer Struktur. Vorgaben sind nicht berücksichtigt. | Die Namensgebung ist verwirrend oder unpräzise. Der Code ist schlecht lesbar. |

1. Individuelles Bewertungskriterium

|  |  |
| --- | --- |
| *Bezeichnung* | |
| 121 - Software-Ergonomie | |
| *Definition (Leitfrage)* | |
| Ist die nötige Benutzerfreundlichkeit/SW-Ergonomie implementiert? | |
| *Gütestufe 3* | *Gütestufe 2* |
| GUI-Elemente sind  -der Funktion entsprechend gewählt  -verständlich beschriftet  -sinnvoll gruppiert und  -korrekt angewendet. | Alle Bewertungspunkte sind weitgehend erfüllt. |
| *Gütestufe 1* | *Gütestufe 0* |
| Es sind bei jedem Bewertungspunkt nur Ansätze vorhanden, oder: es sind ein bis zwei Bewertungspunkte gar nicht erkennbar. | Die Benutzerfreundlichkeit ist mangelhaft. |
|  |  |
|  |  |
|  |  |

1. Individuelles Bewertungskriterium

|  |  |
| --- | --- |
| *Bezeichnung* | |
| 194 - Plausibilisierung der Benutzer-Eingaben | |
| *Definition (Leitfrage)* | |
| Werden die Eingaben des Benutzers überprüft? | |
| *Gütestufe 3* | *Gütestufe 2* |
| Alle Eingabefelder werden überprüft. Es ist eindeutig gekennzeichnet, welche Felder Pflichtfelder sind. Für den Benutzer ist ersichtlich, welche Wertebereiche zulässig sind. Findet die Plausibilisierung eine Fehleingabe, so wird der Benutzer mit konkreten Hinweisen geführt, das entsprechende Feld wird aktiviert. | Plausibilisierung findet statt, Feedback an Benutzer ist mangelhaft/nicht eindeutig/unvollständig. Nur korrekte Daten werden übermittelt. |
| *Gütestufe 1* | *Gütestufe 0* |
| Eingaben werden plausibilisiert, aber bei Fehlern oder fehlenden Eingaben sind die bisher gemachten Eingaben verloren oder die fehlerhaften Eingaben werden trotzdem übermittelt.Oder: es werden nicht alle Eingaben ueberprueft, welche ueberprueft werden sollten. | Es findet keine Plausibilisierung statt. |

### Technologien

* Angular
  + Front-End-Webapplikationsframework
  + Basiert auf Typescript
  + Cross-Plattform
  + Entwickelt von Google
* Express (Node.js)
  + Serverseitiges Webframework
  + Basiert auf JavaScript
* MongoDB
  + Daten werden in JSON-ähnlichen Dokumenten verwaltet
  + No-SQL Datenbankmanagementsystem
  + Flexibel und skalierbar

### Versionierung

Für die Versionierung des Projektes wurde Git verwendet. Dies kann auf der Plattform GitHub unter folgendem Link eingesehen werden:

<https://github.com/seiemens/agenda>

Darauf sind ältere Versionen sowohl vom Code als auch der Dokumentation einzusehen.

## Projektaufbauorganisation

### Projektmitglieder

|  |  |
| --- | --- |
| Name | Rolle |
| Jordan Dainese | Projektmitglied |
| Lara Felix | Expertin |

### Projektmethode

Als Projektmethode kommt IPERKA zum Einsatz.

Die Stärke der IPERKA Methode liegt darin, einen Auftrag Punkt für Punkt zu analysieren und auszuführen. Ausserdem lässt sich damit die Projektphasen klar voneinander trennen. Aufgrund dieser Stärken und persönlichen Erfahrungen aus früheren Modulen haben wir uns somit klar für diese Projektmethode entschieden.

## Zeitplan

### Vorgegebene Termine

|  |  |
| --- | --- |
| Datum | Termin |
| 27.02.2023 | Infotag M223 |
| 14.03.2023 | Projektstart |
| 14.03.2023 | 1. Gespräch mit Experten |
| 22.03.2023 | 2. Gespräch mit Experten |
| 24.03.2023 | Projektabschluss |

### Zeitplan



Abbildung 1 - Zeitplan

## Arbeitsjournal

**14.03.23**

|  |  |
| --- | --- |
| **Geplante Arbeiten** | * Kriterienkatalog studieren * Aufbau der Dokumentation erstellen * Zeitplan erstellen * Expertengespräch |
| **Ausgeführte Arbeiten** | Kriterienkatalog wurde angeschaut. Zeitplan konnte fertig gestellt werden und später auch im Expertengespräch besprochen werden. Nach dem Expertengespräch wurde der Zeitplan noch etwas verbessert. Dokumentation wurde erstellt und Titelblatt, sowie Header- und Footzeilen wurden erstellt. |
| **Hilfestellungen** | Tipps von Lara beim Expertengespräch für Zeitplan und Doku |
| **Erfolge & Misserfolge** | * Zeitplan fertig (ausser soll Zeiten) * Gut in das Projekt gestartet und Zeiten konnten bisher eingehalten werden |
| **Bemerkung** | Meilenstein “Zeitplan fertigstellen” erledigt  1. Gespräch mit Experten fand statt |
| **Nächste Schritte** | Realisierungskonzept erstellen  Aufsetzen der Projektumgebung  Datenbank erstellen  Testkonzept erstellen |
| **Fazit** | Der Anfang des Projekts ist mir so weit geglückt. Ich konnte alles erledigen, was ich mir für heute vorgenommen habe. Da jedoch die Projektzeit sehr kurz ist, kann eine Verzögerung fatal werden und dazu führen, dass ich das Projekt nicht in der vorgegebenen Zeit fertigstellen kann. Als nächstes sollte ich dann mit der Entwicklung am Projekt anfangen. |

**15.03.23**

|  |  |
| --- | --- |
| **Geplante Arbeiten** | * Konzept der Realisierung erstellen * Modell der Datenbank erstellen * Testkonzept erstellen * Projektumgebung einrichten * Dokumentation weiterführen |
| **Ausgeführte Arbeiten** | Da ich mir bis dahin noch nicht 100% sicher war, womit ich mein Projekt umsetzen möchte, musste ich dies zuerst machen. Wir haben am Morgen noch herausgefunden, dass eine relationale Datenbank doch nicht nötig sei, da die Aufgabenstellung, die ich verwendete, noch veraltet war. Aufgrund dessen habe ich mich als Datenbank für MongoDB entschieden, da ich in der Vergangenheit bereits mit dieser gearbeitet habe. Als Frontend, habe ich mich für Angular entschieden und für das Backend ExpressJS. Da ich mit diesen Technologien, besonders Angular, nicht viel Erfahrung hatte musste ich mich zuerst ein wenig einlesen, wie ich überhaupt mein Projekt damit aufsetzen kann. Dies hat aber nach ein wenig probieren funktioniert. Danach habe ich noch die Dokumentation erweitert, und das Arbeitsjournal erstellt. |
| **Hilfestellungen** | - |
| **Erfolge & Misserfolge** | * Dokumentation ist gut vorwärtsgekommen * Datenbank darf mit MongoDB gemacht werden, was mir den Aufwand aufgrund von Erfahrung reduziert * Testkonzept konnte noch nicht erstellt werden, da mir die Zeit fehlte |
| **Bemerkung** | - |
| **Nächste Schritte** | Entwicklung am Projekt anfangen  Dokumentation weiterführen  Testkonzept erstellen |
| **Fazit** | Alles in allem bin ich noch gut in der Zeit. Ich konnte meine Projektumgebung einrichten und meine Datenbank erstellen. Jedoch muss ich mein Testkonzept noch nachholen, da dies noch fehlt. |

**17.03.2022**

|  |  |
| --- | --- |
| **Geplante Arbeiten** | * Datenbank erstellen * Autorisierung (Login und Logout) entwickeln * Übersicht Agenda und Terminpage * Dokumentation weiterführen |
| **Ausgeführte Arbeiten** | Ich habe zuerst meinen Backendserver ausgesetzt und die erste Route fürs Registrieren erstellt, die einen mit gegebenen User in der Datenbank speichert. Dafür habe ich zuerst noch die Verbindung mit MongoDB gemacht. Dann habe ich in meinem Frontend ein Simples Form erstellt, um die Daten für einen User erfasst und dann eine http Request, um die Daten ans Backend zu schicken. Dies hat schliesslich auch nach ein wenig Experimentieren funktioniert und man kann somit einen User registrieren. Da ich bis dahin noch kein git repository hatte, musste ich noch eins erstellen. Dabei bin ich auf ein Problem gestossen. Da ich noch kein git repository initialisiert habe, hat angular beim Erstellen für das Frontend ein Repo initialisiert. Ich habe dies nicht gesehen zudem Zeitpunkt und noch ein git initialisiert, für Front- und Backend und hatte dann sozusagen ein git Verschachtelung. Da ich noch nie eine solche Situation hatte, habe ich dann einen Fehler, in Form eines Git Commands, gemacht, der nicht das git für mein Frontend entfernt, sondern Anfing das Projekt an sich zu löschen. Als ich dies abgebrochen hatte waren schon grosse Teile gelöscht und ich musste das Frontend nochmals von neuem aufsetzen, da ich ja bis dahin noch kein Repository hatte. Danach hat jedoch alles funktioniert und ich konnte mein Repo aufsetzen. Jedoch hat dies unnötig viel Zeit gekostet. |
| **Hilfestellungen** | Arbeitskollegen haben mir beim Einarbeiten mit Angular geholfen |
| **Erfolge & Misserfolge** | * Erste Route im Backend, die einen User in der DB erstellt * Frontend anfrage ans Backend, die Daten für User Erstellung schickt * Viel Zeit verloren beim Initialisieren vom git repository * Entwicklung nicht so weit gekommen wie gehofft, ich bin hinter dem Zeitplan |
| **Bemerkung** | - |
| **Nächste Schritte** | Login route im Backend erstellen  Routing für Angular erstellen  Agenda Pages erstellen  Testkonzept erstellen |
| **Fazit** | Heute war ein Rückschlag. Aufgrund von Unnötigen Zeitverlusten und da ich mich mit Angular nicht gut auskenne, bin ich mit der Entwicklung des Projektes im Verzug. Jedoch konnte ich immerhin die ersten Routes, im Front- und Backend erstellen. Jedoch muss ich auch das Routing von Angular nochmals anschauen und dies Morgen richtig machen. |

**21.03.23**

|  |  |
| --- | --- |
| **Geplante Arbeiten** | * Login route im Backend erstellen * Routing für Angular erstellen * Agenda Pages erstellen * Testkonzept erstellen |
| **Ausgeführte Arbeiten** | Ursprünglich hatte ich mir vorgenommen, heute am Code zu arbeiten. Jedoch bin ich mit dem Projekt im Verzug. Meine Dokumentation hat deutlichen Nachholbedarf. Deshalb habe ich mich dazu entschieden die Priorität auf die Dokumentation zu setzen, da diese bei der Bewertung wichtiger ist. Als erstes habe ich das Testkonzept geschrieben. Danach habe ich den ersten Teil der Doku erweitert, Aufgabenstellung erstellt etc. und der erste Teil der Dokumentation ist jetzt, bis auf vielleicht kleine Zukünftige Änderungen fertig. Danach habe ich noch die Struktur des zweiten Teils der Doku hergestellt. |
| **Hilfestellungen** | Staffbite Dokumentation |
| **Erfolge & Misserfolge** | * Dokumentation ist gut vorwärtsgekommen * Testkonzept konnte abgeschlossen werden * Projekt weit im Verzug, scheint, als werde ich nicht in der Zeit fertig |
| **Bemerkung** | - |
| **Nächste Schritte** | Teil 2 der Doku erweitern  Weiter Routes für das Projekt erstellen  2. Experten Gespräch |
| **Fazit** | Die Dokumentation konnte heute gut weitergeführt werden. Jedoch kann man nicht über den Fakt hinwegsehen, dass ich hinter dem Plan für das Projekt bin. Laut Zeitplan sollte heute die Entwicklung des Projektes fast abgeschlossen sein. Jedoch bin ich nicht annähernd an diesem Punkt. |

**22.03.23**

|  |  |
| --- | --- |
| **Geplante Arbeiten** | * Teil 2 der Doku erweitern * Weiter Routes für das Projekt erstellen * 2. Experten Gespräch |
| **Ausgeführte Arbeiten** | Da ich mit meinem Projekt weiterhin in Verzug bin, habe ich auch heute die Priorität weiterhin auf die Dokumentation gesetzt. Zudem war heute noch das Fachgespräch mit Lara, die mir bestätigte, dass ich meinen Fokus auf eine saubere Dokumentation setzen soll, da dies wichtiger sei. Da der erste Teil der Doku fertig war, habe ich mich an den zweiten Teil gesetzt. Dafür habe ich das Informieren, bis zu dem Entscheiden Teil umgesetzt. Leider bin ich fast nicht mehr zum Entwickeln gekommen, deshalb gibt es in diesem Bereich leider keine Fortschritte. |
| **Hilfestellungen** | Fachgespräch mit Lara |
| **Erfolge & Misserfolge** | * Dokumentation ist gut vorwärtsgekommen * Projekt im Verzug * Keine Zeit mehr für entwickeln gehabt, obwohl erhofft |
| **Bemerkung** | - |
| **Nächste Schritte** | Weiter entwickeln am Projekt, so weit wie möglich  Dokumentation abschliessen |
| **Fazit** | Da ich meinen Fokus auf die Dokumentation lege kommt diese gut voran und scheint realistisch, diese komplett abzuschliessen. Jedoch kann ich dasselbe nicht für |

**24.03.23**

|  |  |
| --- | --- |
| **Geplante Arbeiten (Zeitplan)** | * Weiter entwickeln am Projekt, so weit wie möglich * Dokumentation abschliessen |
| **Ausgeführte Arbeiten** | Heute ist der letzte Tag. Um 11:30 ist Abgabe. Ich habe deshalb noch versucht so viel wie möglich abzuschliessen. Einerseits habe ich am Projekt einiges erweitert, sodass jetzt das Login funktioniert, und man Termine erstellen kann und diese in der Datenbank gespeichert werden. Jedoch kann man diese nicht wieder ausgeben. Zudem habe ich für die Ergonomie ein wenig CSS hinzugefügt. Zudem habe ich die Dokumentation auf den finalen Stand gebracht, heisst R, K und A vom IPERKA hinzugefügt und noch offene Kleinigkeiten abgeschlossen |
| **Hilfestellungen** | - |
| **Erfolge & Misserfolge** | * Projekt Abschluss * Funktionalitäten müssen unvollständig abgegeben werden |
| **Bemerkung** | - |
| **Nächste Schritte** | Präsentation und Fachgespräch |
| **Fazit** | Ich bin einerseits zufrieden mit meinem Projekt, vor allem mit der Dokumentation, dass die fertig geworden ist. Ich hoffe für das nächste Mal, dass ich auch mit dem Code fertig werde. |

# Projekt (Teil 2)

## Kurzzusammenfassung

### Ausgangslage

Das Ziel ist, eine Weblösung für eine Agenda zu erstellen. Dabei kann ein User beliebig viele Einträge in seine Agenda erstellen, die er mit Titel, allenfalls Beschreibung und einem Datum versehen kann. Zudem sollte er erstellte Einträge editieren und auch wieder löschen können. Sobald ein User Termine erstellt hat, erscheinen ihm diese auf seiner Übersicht Seite, wodurch er eine Übersicht über alle Termine bekommt. Ein Adminuser hat zusätzlich Zugriff auf ein Adminpanel, mit dem er Benutzer verwalten kann.

### Umsetzung

Das Projekt wurde unter Verwendung der IPERKA-Projektmethodik durchgeführt.

Als Erstes stand die Informationsphase an, in dieser wurde gemäss der Aufgabestellung Anforderungen definiert.

Während der Planungsphase wurde ein Zeitplan, eine Grundstruktur für die Dokumentation erstellt.

In der Realisierungsphase wurde mithilfe des Angular-Webframework basierend auf Typescript, HTML und CSS das Frontend und mithilfe von Express basierend auf Node.js das Backend entwickelt.

Die Kontrollphase bestand aus manuellen Tests.

Zum Schluss stand eine Auswertung an, in der Erfolge, Probleme und Verbesserungsmöglichkeiten analysiert wurden. Aufgrund der vorliegenden Fakten wurde ein Fazit gezogen.

## Informieren

Das genaue Analysieren der Aufgabenstellung ist das Wichtigste in der ersten Phase der Projektplanungsmethode IPERKA.

### Ziele der Aufgabenstellung

In der Applikation kann jeder Benutzer, der angemeldet ist mit seiner Persönlichen Agenda interagieren. Dabei kann er persönliche Einträge erstellen, bearbeiten und löschen.

Ein Benutzer mit der Administrator Rolle hat zusätzlich Zugriff auf ein Adminpanel, bei dem er Benutzer verwalten kann.

Das Resultat wird als Webapplikation bereitgestellt, damit man es leicht hosten und weiter verwenden kann. Somit werden die Daten zentral gespeichert.

## Planen

In dieser Phase wird der Verlauf des Projektes bestimmt und ist somit ein essenzieller Teil in der Projektplanungsmethode IPERKA. In dieser Phase wird der zeitliche Ablauf bestimmt und in den Zeitplan eingefügt.

Der Zeitplan befindet sich beim Punkt «Zeitplan». Aus Redundanzgründen wird dieser hier nicht nochmals abgebildet.

Das Projekt wurden von Anfang bis Schluss mit der IPERKA Projektplanungsmethode geplant und durchgeführt.

### Sitemap

* **Login**
  + Hier kann ein Nutzer sich in seinen Account einloggen
* **Register**
  + Ein neuer Benutzer kann hier seinen Account anlegen
* **Home**
  + Übersicht über alle persönlichen Termine
  + Möglichkeit zum Termine löschen

1. Edit Meeting
   * Möglichkeit zum Termine bearbeiten

* **User Management** (Nur für Admin Benutzer)
  + Benutzer erstellen, bearbeiten und löschen

### Testkonzept

|  |  |
| --- | --- |
| Testfall Nr. 1 | User Register |
| Beschreibung | Ein Benutzer versucht sich zu registrieren |
| Durchführung | * Ein Benutzer gibt auf der Registrieren Seite seine Eingaben ein und drückt auf den Knopf |
| Erwartetes Resultat | Er wird zur Login Page weitergeleitet und ein Nutzer wird in der Datenbank angelegt |

|  |  |
| --- | --- |
| Testfall Nr. 2 | User Register Empty |
| Beschreibung | Ein Benutzer versucht sich zu registrieren |
| Durchführung | * Ein Benutzer gibt auf der Registrieren Seite seine Eingaben ein, jedoch wird ein, oder mehrere Fehler leer gelassen, und drückt auf den Knopf |
| Erwartetes Resultat | Eine Fehlermeldung taucht auf, die ihn auf die leeren Felder hinweist |

|  |  |
| --- | --- |
| Testfall Nr. 3 | User Login existent |
| Beschreibung | Ein Benutzer versucht sich anzumelden |
| Durchführung | * Ein Benutzer gibt auf der Login Seite seine korrekten Eingaben ein und drückt auf den Knopf |
| Erwartetes Resultat | Er wird authentifiziert und kommt auf seine Übersichtpage mit seinen Terminen |

|  |  |
| --- | --- |
| Testfall Nr. 4 | User Login non existent |
| Beschreibung | Ein Benutzer gibt falsche Eingaben für das Login ein |
| Durchführung | * Ein Benutzer gibt auf der Login Seite Eingaben ein, die nicht existieren und drückt auf den Knopf |
| Erwartetes Resultat | Eine Fehlermeldung wird angezeigt und er kann es nochmal probieren |

|  |  |
| --- | --- |
| Testfall Nr. 5 | User Create Meeting |
| Beschreibung | Ein Benutzer erstellt einen neuen Termin für seine Agenda |
| Durchführung | * Ein Benutzer drückt auf der Übersichtpage den Button zum Termin erstellen * Er wird weitergeleitet auf eine Page wo er den Termin Titel, Beschreibung und Datum eingibt * Der Benutzer drückt auf Bestätigen |
| Erwartetes Resultat | Der Benutzer wird zur Übersichtseite weitergeleitet und sieht da seinen neu erstellten Termin |

|  |  |
| --- | --- |
| Testfall Nr. 6 | User Create Meeting Past |
| Beschreibung | Ein Benutzer erstellt einen neuen Termin für seine Agenda, jedoch ist das Datum in der Vergangenheit |
| Durchführung | * Ein Benutzer drückt auf der Übersichtpage den Button zum Termin erstellen * Er wird weitergeleitet auf eine Page wo er den Termin Titel, Beschreibung und Datum eingibt, jedoch ist das Eingegebene Datum bereits vergangen * Der Benutzer drückt auf Bestätigen |
| Erwartetes Resultat | Der Benutzer bekommt eine Fehlermeldung, die ihn darauf hinweist, dass sein Datum bereits vergangen ist. |

|  |  |
| --- | --- |
| Testfall Nr. 7 | User Create Meeting Empty |
| Beschreibung | Ein Benutzer erstellt einen neuen Termin, jedoch ist Titel und Datum nicht ausgefüllt |
| Durchführung | * Ein Benutzer drückt auf der Übersichtpage den Button zum Termin erstellen * Er wird weitergeleitet auf eine Page, wo er den Termin Titel, Beschreibung und Datum eingeben kann, jedoch alles leer lässt * Der Benutzer drückt auf Bestätigen |
| Erwartetes Resultat | Ein Fehler Meldung taucht auf, die ihm die Felder markiert, welche ihn auf die fehlenden Eingaben hinweist |

|  |  |
| --- | --- |
| Testfall Nr. 8 | User Create Meeting Empty Description |
| Beschreibung | Ein Benutzer erstellt einen neuen Termin, jedoch ist die Beschreibung nicht ausgefüllt |
| Durchführung | * Ein Benutzer drückt auf der Übersichtpage den Button zum Termin erstellen * Er wird weitergeleitet auf eine Page, wo er den Termin Titel, Beschreibung und Datum eingeben kann, er füllt aber nur Titel und Datum aus * Der Benutzer drückt auf Bestätigen |
| Erwartetes Resultat | Der Benutzer wird zur Übersichtseite weitergeleitet und sieht da seinen neu erstellten Termin ohne Beschreibung |

|  |  |
| --- | --- |
| Testfall Nr. 9 | User Edit Meeting |
| Beschreibung | Ein Benutzer editiert einen vorhandenen Termin für seine Agenda |
| Durchführung | * Ein Benutzer drückt auf der Übersichtpage den Editierbutton, der bei einem Termin steht * Er wird weitergeleitet auf eine Page wo er bei dem Termin Titel, Beschreibung und Datum anpassen kann * Der Benutzer drückt auf Bestätigen |
| Erwartetes Resultat | Der Benutzer wird zur Übersichtseite weitergeleitet und sieht da seinen Termin mit den Änderungen |

|  |  |
| --- | --- |
| Testfall Nr. 10 | User Delete Meeting |
| Beschreibung | Ein Benutzer löscht einen vorhandenen Termin für seine Agenda |
| Durchführung | * Ein Benutzer drückt auf der Übersichtpage den Deletebutton, der bei einem Termin steht |
| Erwartetes Resultat | Der Termin wird aus der Datenbank gelöscht und somit auch von der Übersichtseite entfernt |

|  |  |
| --- | --- |
| Testfall Nr. 11 | Admin Create User |
| Beschreibung | Ein Admin Benutzer erstellt im Adminpanel einen Benutzer |
| Durchführung | * Ein Admin klickt bei der Navigation, auf Admin * Er kommt auf die Seite für die Benutzerverwaltung * Er drückt auf den Erstellen button * Er wird weitergeleitet auf eine Seite, um die Eingaben für den Benutzer zu tätigen * Er füllt die Felder aus und drückt auf Bestätigen |
| Erwartetes Resultat | Ein Benutzer wird in der Datenbank angelegt |

|  |  |
| --- | --- |
| Testfall Nr. 12 | Admin Edit User |
| Beschreibung | Ein Adminbenutzer editiert im Adminpanel einen Benutzer |
| Durchführung | * Ein Admin klickt bei der Navigation, auf Admin * Er kommt auf die Seite für die Benutzerverwaltung * Er drückt auf den editieren Button bei einem Nutzer * Er wird weitergeleitet auf eine Seite, um die Eingaben für den Benutzer zu editieren * Er füllt die Felder aus und drückt auf Bestätigen |
| Erwartetes Resultat | Die Daten von dem editierten Benutzer werden in der Datenbank geändert |

|  |  |
| --- | --- |
| Testfall Nr. 13 | Admin Delete User |
| Beschreibung | Ein Adminbenutzer löscht im Adminpanel einen Benutzer |
| Durchführung | * Ein Admin klickt bei der Navigation, auf Admin * Er kommt auf die Seite für die Benutzerverwaltung * Er drückt auf den löschen Button bei einem Nutzer * Er bestätigt bei der Meldung seine Aktion |
| Erwartetes Resultat | Der Benutzer wird aus der Datenbank gelöscht |

## Entscheiden

In der dritten Phase der IPERKA Projektplanungsmethode wird entschieden, welcher Lösungsweg verwendet wird. Dabei werden Lösungsvarianten verglichen und allfällige Risiken evaluiert.

### Varianten

Vor dem Anfang muss man zuerst entscheiden, welches Framework und welche verwendet werden soll. Aufgrund des kleinen Zeitfensters kam ein komplett neues Framework und eine neue Datenbank zu verwenden nicht wirklich infrage.

Für das Frontend gab es die Optionen Vue, Nuxt oder Angular. Für die Datenbank standen SQL und MongoDB infrage. Als Backend .NET Core oder ExpressJS.

### Risiken

Das Risiko mit einer SQL-Datenbank war, dass ich bereits eine Weile nicht mehr damit gearbeitet habe und nicht weiss, wie man es ins Projekt einbinden kann. Es war klar das SQL jedoch eine bessere Variante für das Speichern von persönlichen Daten wäre.

Mit der MongoDB hatte ich bereits Erfahrung und wusste, dass persönliche Daten genauso gut in einer NoSQL-DB gespeichert werden können. Zudem würde mir dies wahrscheinlich Aufgrund der Erfahrung leichter fallen.

Beim Frontend war die Entscheidung schwierig für mich, da ich bereits eine Weile nicht mehr ein Frontend entwickelt habe und auch in meiner Abteilung zurzeit nur Backend arbeite. Ich habe mir deshalb bei den anderen Lernenden ein paar Ideen geholt, was ich verwenden könnte.

### Entscheid

Aufgrund von Vorkenntnissen habe ich mich für eine MongoDB Datenbank entschieden. Da ich zurzeit in meiner Abteilung mit NodeJS arbeite habe ich mich für das ExpressJS Backend entschieden. Beim Frontend habe ich mich für Angular entschieden, weil ich damit vor einer Weile mal gearbeitet habe und deshalb immerhin ein wenig Erfahrung aufweisen kann. Zudem haben viele der anderen Lernenden Erfahrungen damit und ich könnte somit leicht nach Rat fragen, wenn ich feststecke.

## Realisieren

### Klassendiagramm

Unten abgebildet ist mein Klassendiagramm. Dies habe ich nicht im Vorhinein geplant, jedoch ist es recht früh während der Realisierung entstanden.

Ein Bild, das Diagramm enthält.

Automatisch generierte Beschreibung

Abbildung 2 - Klassendiagramm

### Umsetzungen

Da ich mit meinem Projekt nicht fertig wurde, Teile ich den Realisieren Teil in «Umsetzungen», bei dem ich die Sachen beschreibe, die ich Umsetzen konnte, und «Nicht Umgesetzt», bei dem ich Teile beschreibe, die ich geplant hatte, jedoch aus zeitlichen Gründen nicht mehr umsetzen konnte, auf.

#### Backend

##### Verbindung zur Datenbank

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungMeine verwendete Datenbank ist MongoDB. Die Verbindung konnte dank dem Node Modul Mongoose konnte die Verbindung zur Datenbank sehr einfach hergestellt werden. Anbei noch ein Screenshot.

Abbildung 3 - Datenbankverbindung

##### Ein Bild, das Text enthält. Automatisch generierte BeschreibungModels

Abbildung 4 - Backendmodel

Auch für die verwendeten Models konnte mir das Node Modul mongoose unter die Arme greifen. Dies sieht wie folgt aus:

Mit Mongoose kann man sogenannte Schemas erstellen. Bei einem Schema definiert man die Felder, die man abspeichern möchte. Bei diesem Beispiel wäre das email, username und password für einen Benutzer. Danach kann man noch die Collection, also wo in der MongoDB man diese Schemas speichern möchte, angeben und fertig ist ein Model.

##### Routes

Ein Bild, das Text, Screenshot, Im Haus, Bildschirm enthält.

Automatisch generierte BeschreibungFür die Routes, die die Requests entgegennimmt, habe ich ExpressJS verwendet. Als Beispiel verwende ich hier die Route zum Registrieren. Da das registrieren eine POST request ist verwendet man app.post(). App ist mein ExpressJS Server der läuft. Als Argumente für diese Funktion gebe ich den Pfad, auf den die Request hört, und eine Callback Funktion, die ausgeführt wird, wenn eine Request ankommt. Für das registrieren erstelle ich dafür mit dem mongoose Schema einen neuen Nutzer, und lese die Daten aus der Request aus. Mit der .save() Funktion wird dieser Nutzer gleich in der vom Schema angegebenen Collection in der MongoDB gespeichert. Danach wird eine Antwort zurück geschickt.

Abbildung 5 - Backend Route

#### Frontend

##### Routing

In Angular kann man das Wechseln zwischen den Komponenten mit einem Routing Module lösen. Dies ist in Angular integriert. Man kann seine Routes definieren, indem man einen Pfad angibt und die jeweilige Komponente, der dann aufgerufen werden soll. Anbei ein Beispiel mit dem Login Route:

Abbildung 6 - Frontend Route

##### Komponente

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungEin Komponent in Angular besteht aus einem HTML-File, einem SCSS-File und einem Code-Behind in Form eines TypeScript Files. Ich beschreibe als Beispiel noch, wie der Login Komponente funktioniert. Im HTML-File ist ein simples Form für den Username und das Passwort des Benutzers. Im TypeScript File hat es eine Funktion, die ausgeführt wird, sobald der Knopf des Forms gedrückt wird. Dabei wird geprüft, ob der Benutzer eingaben getätigt hat und falls ja eine Anfrage über den UserService ans Backend geschickt, der die Anfrage entgegennimmt und in der Datenbank nach dem Benutzer sucht. Als Antwort kommt entweder der Vollständige Benutzer mit einem Code 200, oder ein 404 Not Found Fehler.

Abbildung 7 - Frontend Login Funktion

##### Service

In meinen Services sind Funktionen, die ich bei mehreren Komponenten verwenden möchte. Beispielsweise sind in meinem UserService die Anfragen ans Backend definiert. Dabei wird die richtige URL zusammengebaut und eine Anfrage geschickt.

### Nicht umgesetzt

#### Adminpanel

Das erste Feature, auf das ich verzichten müsse, ist das Adminpanel, bei dem ein Admin Benutzer Zugriff auf eine Benutzerverwaltung hat. Dies hätte ich eingebaut, indem ich das Model für den User erweitert hätte mit einem weiteren Feld. Dies wäre Berechtigung und hat entweder den Wert 0 oder 1. Ein Nutzer mit dem Wert 1 hat dann Zugriff auf das Adminpanel. Dieses wäre, wenn die Berechtigung da ist, ein weiterer Punkt bei der Navigation gewesen. Es würden alle Nutzer aufgelistet werden und der Admin kann entweder mit einem Button einen neuen erstellen, oder die jeweiligen Benutzer editieren und auch löschen. Den Aufwand dafür hätte ich etwa auf 2h eingeschätzt.

## Kontrollieren

### Manuelle Tests

Aufgrund der fehlenden Zeit konnte ich meine Tests des Testkonzepts nicht mehr ausführen.

## Auswerten

Das Auswerten ist der letzte Schritt der IPERKA Projektplanungsmethode. In diesem Teil wird reflektiert, Erfolge und Misserfolge erkannt, sowie Verbesserungsmöglichkeiten gesucht und zum Schluss ein Fazit daraus gezogen.

## Reflexion

### Erfolge

Der grösste Erfolg für mich war, dass ich wertvolle Erfahrungen für die richtige IPA sammeln konnte. Ich konnte mich weiterentwickeln und weiss jetzt ungefähr, wie meine Abschlussarbeit ablaufen wird. Zudem konnte ich meine Kenntnisse mit Angular ein wenig auffrischen, was mir zukünftige Arbeit damit wahrscheinlich erleichtern wird.

### Misserfolge

Leider war dieses Projekt kein voller Erfolg. Ich konnte nicht alle Funktionalitäten für meine Applikation erstellen, die ich mir vorgenommen hatte. Zudem ist mir ein Fehler mit git passiert, der mich noch zusätzlich 1-2 h gekostet hat.

### Verbesserungsmöglichkeiten

Was ich mir am meisten vornehmen werde für die nächste Arbeit mit einer so begenzten Zeitspanne ist die Vorbereitung. Ich bin mit dem Projekt nicht so schnell vorangekommen, da mir vor allem die Erfahrung mit Angular fehlt. Für das nächste mal sollte ich mich vorher besser in ein unbekanntes Framework einarbeiten, um während des Projektes eine höhere Effizienz zu haben.

### Fazit

Alles in allem war das Projekt eine gute Erfahrung für mich. Ich kann über die Misserfolge hinweg sehen, weil ich weiss, was ich das nächste Mal besser machen kann, um eine solche Situation zu verhindern. Ich denke das nächste Mal werde ich auch mehr Prozent der Funktionalitäten abschliessen können.

# Quellenverzeichnis

* Angular: <https://angular.io/>
* IPERKA: <https://www.bexio.com/de-CH/blog/view/iperka-methode>
* Mongo DB: <https://www.mongodb.com/home>
* Staffbite: <https://github.com/dar1ooo/staffbite>
* ChatGPT: <https://chat.openai.com/>

# Glossar

**API** -> Eine API (Application Programming Interface) ist eine Schnittstelle, die ermöglicht, dass verschiedene Computerprogramme miteinander kommunizieren und Daten und Funktionen austauschen können.

**Sitemap** -> Die Sitemap soll beim Realisieren einen guten Überblick über die verschiedenen Seiten und deren Relation verschaffen. Sie zeigt somit die logische Struktur der Webapplikation auf.

**Client** -> Der Client ist ein Gerät, welches mit einem Netzwerk verbunden ist. Er sendet Anfragen an den Server und der Server liefert die gewünschten Daten oder Dienste zurück.

**Callback** -> Eine Funktion, die einer anderen Funktion als Parameter mitgegeben wird.

**TypeScript** -> TypeScript ist eine Programmiersprache, die auf JavaScript basiert. Bei laufzeit wird diese zu JavaScript kompiliert, damit der Browser das Skript verstehen kann.

**ExpressJS** -> Express ist ein auf JavaScript basierendes Webframework, womit man Webserver aufsetzen und konfigurieren kann

**Angular** -> Angular ist ein Frontendframework, das auf TypeScript basiert.

# Abbildungsverzeichnis

[Abbildung 1 - Zeitplan 9](#_Toc130548569)

[Abbildung 2 - Klassendiagramm 22](#_Toc130548570)

[Abbildung 3 - Datenbankverbindung 22](file:///C:\Users\jad_2\Documents\M223\Agenda\documentation\Probe%20IPA%20Dokumentation.docx#_Toc130548571)

[Abbildung 4 - Backendmodel 23](file:///C:\Users\jad_2\Documents\M223\Agenda\documentation\Probe%20IPA%20Dokumentation.docx#_Toc130548572)

[Abbildung 5 - Backend Route 23](file:///C:\Users\jad_2\Documents\M223\Agenda\documentation\Probe%20IPA%20Dokumentation.docx#_Toc130548573)

[Abbildung 6 - Frontend Route 23](file:///C:\Users\jad_2\Documents\M223\Agenda\documentation\Probe%20IPA%20Dokumentation.docx#_Toc130548574)

[Abbildung 7 - Frontend Login Funktion 24](file:///C:\Users\jad_2\Documents\M223\Agenda\documentation\Probe%20IPA%20Dokumentation.docx#_Toc130548575)

# Anhang

## Quellcode

### /agenda-backend/app.js

const express = require('express');

const app = express();

const mongoose = require('mongoose');

const bodyParser = require('body-parser');

const cors = require('cors');

User = require('./models/user');

Meeting = require('./models/meeting');

require('dotenv/config');

app.use(bodyParser.json());

app.use(cors());

//connect to mongodb via url

mongoose.connect("mongodb://localhost:27017/mydb",

  { useNewUrlParser: true, useUnifiedTopology: true })

    .then(() => console.log('Connected to MongoDB!'))

    .catch(err => console.error('Error connecting to MongoDB:', err));

//check if user exists in db

app.post('/api/login', (req, res) => {

    const requestData = req.body;

    User.findOne({ username: req.body.username, password: req.body.password })

        .then((user) => {

        if (!user) {

          // User not found

          res.status(404).send();

          console.log('User not found');

        } else {

          // User found

          res.json(user);

          console.log('User found:', user);

        }

      })

      .catch((err) => {

        console.log(err);

      })

    });

app.post('/api/register', (req, res) => {

    //create new user with mongoose schema

    const newUser = new User({username: req.body.username, email: req.body.email, password: req.body.password})

    //save user to db

    newUser.save()

    .then(() => console.log('Data saved to MongoDB!'))

    .catch(err => console.error('Error saving data to MongoDB:', err));

    //send response

    const responseData = { message: 'Created User' };

    res.json(responseData);

});

app.post('/api/meeting', (req, res) => {

const meeting = new Meeting({title: req.body.title, description: req.body.description, startTime: req.body.startDate, endTime: req.body.endDate})

meeting.save()

});

//start webserver

var port = process.env.PORT || '3000'

app.listen(port, err => {

  if (err)

    throw err

  console.log('Server listening on port', port)

})

### /agenda-backend/user.js

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

  email: { type: String, required: true, unique: true },

  username: { type: String, required: true, unique: true },

  password: { type: String, required: true },

},  { collection: 'user' });

module.exports = mongoose.model('User', userSchema);

### /agenda-frontend/src/app/app-routing.module.ts

import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { CreateComponent } from './components/create/create.component';

import { DashboardComponent } from './components/dashboard/dashboard.component';

import { LoginComponent } from './components/login/login.component';

import { RegistrationComponent } from './components/registration/registration.component';

const routes: Routes = [

  { path: 'login', component: LoginComponent },

  {path: 'register', component: RegistrationComponent},

  {path: 'home', component: DashboardComponent},

  {path: 'create', component: CreateComponent},

  { path: '\*\*', redirectTo: '/' }

];

@NgModule({

  imports: [RouterModule.forRoot(routes)],

  exports: [RouterModule]

})

export class AppRoutingModule { }

### /agenda-frontend/src/app/components/login/login.component.ts

import { Component } from '@angular/core';

import { UserService } from 'src/app/services/user.service';

import { UserLogin } from 'src/app/models/userLogin.model';

import { catchError, tap } from 'rxjs';

@Component({

  selector: 'app-login',

  templateUrl: './login.component.html',

  styleUrls: ['./login.component.scss']

})

export class LoginComponent {

  public userlogin = new UserLogin();

  constructor(

    private userService: UserService

  ){}

  public login(): void {

    if(this.userlogin.password !== '' && this.userlogin.username !== ''){

      this.userService.loginUser(this.userlogin).pipe(

        tap((res) =>{

          console.log(res)

          sessionStorage.setItem("username", res.username)

          window.location.href = '/home'

        }),

        catchError((err) =>{

          return err;

        })

      ).subscribe();

    }

  }

}

### /agenda-frontend/src/app/services/user.service.ts

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { UserRegister } from '../models/userRegister.model';

import { Observable } from 'rxjs';

import { UserLogin } from '../models/userLogin.model';

import { Meeting } from '../models/meeting.model';

@Injectable({

  providedIn: 'root'

})

export class UserService {

  constructor(private http: HttpClient) {}

  private baseurl = 'http://localhost:3000/api/';

  registerUser(userSignUp: UserRegister): Observable<any> {

    const url = this.baseurl + 'register';

    return this.http.post<UserRegister>(url, userSignUp);

  }

  loginUser(userLogin: UserLogin): Observable<any> {

    const url = this.baseurl + 'login';

    return this.http.post<UserLogin>(url, userLogin);

  }

  createMeeting(meeting: Meeting): Observable<any> {

    const url = this.baseurl + 'meeting';

    return this.http.post<UserLogin>(url, meeting);

  }

}

### /agenda-frontend/src/app/components/register/register.component.ts

import { Component } from '@angular/core';

import { tap } from 'rxjs';

import { UserRegister } from 'src/app/models/userRegister.model';

import { UserService } from 'src/app/services/user.service';

@Component({

  selector: 'app-registration',

  templateUrl: './registration.component.html',

  styleUrls: ['./registration.component.scss']

})

export class RegistrationComponent {

  email: string | undefined;

  username: string | undefined;

  password: string | undefined;

  constructor(

    private userService: UserService

  ){}

  onSubmit() {

    if(this.email && this.username && this.password){

      var user: UserRegister = {username: this.username, email: this.email, password: this.password};

      this.userService.registerUser(user).subscribe();

      console.log("did req");

    }

    }

}

### /agenda-frontend/src/app/components/register/register.component.html

<form class="registration-form" (ngSubmit)="onSubmit()">

  <h1>Registration Form</h1>

  <div>

    <label for="email">Email</label>

    <input type="email" name="email" [(ngModel)]="email">

  </div>

  <div>

    <label for="username">Username</label>

    <input type="text" name="username" [(ngModel)]="username">

  </div>

  <div>

    <label for="password">Password</label>

    <input type="password" name="password" [(ngModel)]="password">

  </div>

  <button type="submit">Register</button>

</form>

### /agenda-frontend/src/app/components/create/create.component.ts

import { Component } from '@angular/core';

import { Meeting } from 'src/app/models/meeting.model';

import { UserService } from 'src/app/services/user.service';

@Component({

  selector: 'app-create',

  templateUrl: './create.component.html',

  styleUrls: ['./create.component.scss']

})

export class CreateComponent {

  title: string | undefined;

  description: string | undefined;

  start: Date | undefined;

  end: Date | undefined;

  constructor(

    private userService: UserService

  ){}

  onSubmit() {

    if(this.title != undefined && this.description != undefined && this.start != undefined && this.end != undefined){

      var meeting: Meeting = {title: this.title, description: this.description, endDate: this.end, startDate: this.start}

      this.userService.createMeeting(meeting).subscribe();

      window.location.href = '/home';

    }

    console.log(this.title, this.description, this.start, this.end);

  }

}