

Hochschule Osnabrück
University of Applied Sciences

Fakultät
Ingenieurwissenschaften und Informatik

Schriftliche Ausarbeitung zum Thema:

**Erstellung einer HTML5-/CSS-/Javascript-App unter Verwen-
dung von Capacitor**

im Rahmen des Moduls
Webanwendung,
des Studiengangs Informatik

Autor: Louis Schröder
Matr.-Nr.: 863769
E-Mail: louis.schraeder@hs-osnabrueck.de

Autor: Benno Steinkamp
Matr.-Nr.: 855624
E-Mail: benno.steinkamp@hs-osnabrueck.de

Themensteller: Dipl.-Inf. (FH) Björn Plutka

Abgabedatum: 24.09.2021

Inhaltsverzeichnis

Inhaltsverzeichnis	2
Abbildungsverzeichnis	3
1 Einleitung	1
1.1 Vorstellung des Themas	1
1.2 Ziel der Ausarbeitung	1
1.3 Aufbau der Hausarbeit	1
2 Darstellung der Grundlagen	2
2.1 Node.js	2
2.2 Ionic 2	
2.3 Angular	2
2.4 Capacitor	2
2.5 ZXING/NGX-Scanner	2
2.5.1 QR-Code	2
3 Anwendung	4
3.1 Rest-API	4
3.2 Login	5
3.3 Plan Verwaltung	6
3.4 Möbel Verwaltung	8
3.5 Editor	11
4 Ausblick	12
4.1 Erweiterung	12
4.2 Kommerzielle Nutzung	12
5 Zusammenfassung und Fazit	13
6 Literaturverzeichnis	14

Abbildungsverzeichnis

Abbildung 1: Aufbau QR-Code.....	3
Abbildung 2: Login Page.....	5
Abbildung 3: Register Page	5
Abbildung 4: Tab Plan Verwaltung.....	6
Abbildung 5: Plan erstellen	7
Abbildung 6: Plan ändern.....	7
Abbildung 7: Slider.....	7
Abbildung 8: Tab Möbelverwaltung	8
Abbildung 9: Modal Möbel ändern.....	9
Abbildung 10: Modal Möbel erstellen	9
Abbildung 11: Ionic Fab-Button.....	9
Abbildung 12: Modal QR-Code scanne.....	10
Abbildung 13: QR-Code Scanner.....	10
Abbildung 14 Editor	11

1 Einleitung

Jeder kennt das Problem bei einem Umzug, zuerst müssen die gesamten Räume ausgemessen werden, um zu überprüfen, ob und welche Möbel in ein Raum passen. Oft kommt es vor, dass beim Aufbauen doch nicht alle Möbel in den Raum passen, da kein ordentlicher Plan erstellt wurde. Mit diesem Projekt wollen wir für das Problem eine Lösung erarbeiten, die es ermöglicht schon im Vorhinein einen Umzug sauber zu Planen.

1.1 Vorstellung des Themas

Die Aufgabe des Projektes ist es, einen Room-planer-App zu entwickeln. Diese App soll es den Nutzern ermöglichen, Pläne für neue Räume zu erstellen, um schon im Voraus zu prüfen, ob Möbel in den Raum passen oder nicht. Hierzu soll ein Editor erstellt werden, in dem der Nutzer, seine Möbel frei bewegen kann, dadurch soll der Nutzer in der Lage sein, zusehen wo es zu Überschneidungen der Möbel kommt.

Darüber hinaus soll der Nutzer die Möglichkeit haben Räume und Möbel zu erstellen und zu verwalten. Dabei besitzen jeder Raum und jedes Möbelstück eine durch den Nutzer festgelegte Größe, bei den Möbelstücken kann des Weiteren auch eine Farbe festgelegt werden, durch die sie in den Plänen identifiziert werden können.

1.2 Ziel der Ausarbeitung

Das Ziel des Projektes ist es eine App, auf Basis von Ionic zu entwickeln, hierbei liegt der Fokus auf einer intuitiven Bedienung und der Funktionalität der einzelnen Komponenten. Als backend soll die Rest-API von Benno Steinkamp und Christoph Freimuth eingesetzt werden, diese ermöglicht es uns eigene Pläne, sowie Möbel zu erstellen. Darüber hinaus lassen sich einem Plan mehrere Möbel zuordnen, mit einer bestimmten Position.

Dem Nutzer soll die Möglichkeit geboten werden Pläne und Möbel zu erstellen, bearbeiten und zu löschen. Der Editor soll so gestaltet werden, dass die Vorher erstellten Möbel eingefügt werden können und der Nutzer diese so anordnen kann wie es am besten passt.

1.3 Aufbau der Hausarbeit

Zu Beginn der Hausarbeit gehen wir genauer auf die verwendeten Frameworks ein, die im Projekt verwendet wurden. Im Anschluss gehen wir dann auf die konkrete Umsetzung ein, dabei fokussieren wir uns auf die Funktionen der App und welche Komponenten zum Einsatz kommen.

Bei der Umsetzung beginnen wir damit, die Rest-API genauer zu beschreiben, genauer gesagt beschreiben wir welche Endpunkte wir verwendet haben und wie diese angesprochen werden. Im Anschluss gehen wir Schritt für Schritt durch die App und zeigen dabei, welche Funktionen sie bietet und mit welchen Komponenten diese Funktion umgesetzt wurde. Zum Ende wollen wir noch einen Ausblick geben, um welche Komponenten die App erweitert werden kann und wie die App z.B. in einem Möbelhaus eingesetzt werden kann.

2 Darstellung der Grundlagen

2.1 Node.js

Node.js ist eine JavaScript Runtime basierend auf der V8 JavaScript Engine, welche auch im Chrome Browser verwendet wird. Zur Node.js Umgebung gehört auch npm der Node Package Manager, welcher die Dependencyverwaltung übernimmt.

2.2 Ionic

Ionic ist ein Open Source Toolkit, um Webanwendungen plattformunabhängig im nativen Stil der jeweiligen Plattform zu entwickeln. Es bietet Integration mit den Frameworks Angular, Vue und React an. Und stellt dort Komponenten zur Verfügung, welche bestimmte, für Mobile Anwendungen bekannte Funktionalitäten nachahmen bzw. implementieren. Des Weiteren wird die Verwendung verschiedener Native Ressourcen ermöglicht (Kamera, Kontakte, etc.). Das Ionic CLI integriert außerdem sowohl Angular, Vue und React, als auch Capacitor.

2.3 Angular

Angular ist ein Framework zur Entwicklung von progressiven Front-End-Webapps basierend auf Node.js. Des Weiteren ist Angular grundsätzlich TypeScript basiert. TypeScript ist eine Scriptsprache, welche zu JavaScript transpiliert wird. Angular Anwendungen sind komponentenbasiert aufgebaut. Komponenten bestehen dabei einerseits aus HTML-Code, welcher weitere Komponenten enthalten kann und andererseits aus TypeScript Code, welcher die Funktionalität der Komponente bestimmt. Außerdem können Services erzeugt werden, welche per Dependency Injection verwendet werden können, um weitere Funktionalität anzubieten. Angular verfügt über eine weite Auswahl von Bibliotheken.

2.4 Capacitor

Capacitor ist eine Open Source basierte Runtime für Webapps auf mobilen Endgeräten (hauptsächlich IOS und Android).

2.5 ZXING/NGX-Scanner

Bei dem ZXING/NGX-Scanner handelt es sich um einen Barcode/ QR-Code Scanner, der in Angular eingebunden werden kann. Das Ganze ist ein Git-Hub Projekt, das von mehreren Personen aus der ganzen Welt weiterentwickelt wird. In unserem Projekt wurde die Funktion des QR-Code Scanners benutzt.

2.5.1 QR-Code

Bei dem QR-Code handelt es sich um einen zweidimensionalen Code. Die Informationen werden dabei, in waagerechte und senkrechte Richtung geschrieben. Im Gegensatz dazu wird beim Barcode nur auf einer Dimension geschrieben, und zwar aus der waagerechten Richtung.

Ein Problem beim zweidimensionalen gegenüber dem eindimensionalen Code ist laut [1], dass lokalisieren des Codes, denn dadurch wird es erheblich langsamer. Um das Problem zu umgehen, wurden beim QR-Code sogenannte „Position detection pattern“

eingesetzt, dabei handelt es sich um ein einzigartiges Symbol, dass in drei Ecken des QR-Codes verteilt werden. In der unteren Abbildung ist gezeigt an welcher Position die Symbole stehen.

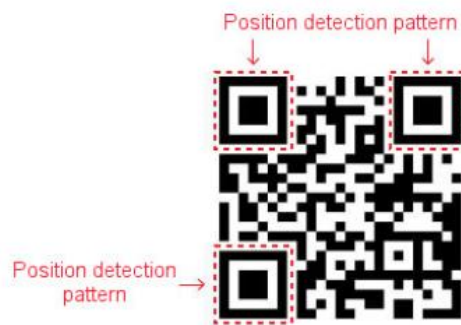


Abbildung 1: Aufbau QR-Code

3 Anwendung

In diesem Kapitel wollen wir auf die einzelnen Komponenten unserer App eingehen, und diese genauer erklären. Dabei starten wir mit der Rest-API, denn auf deren Basis wurde die App aufgebaut. Im Anschluss werden wir nach und nach durch das Menü gehen, dabei werden wir genauer auf Funktionen eingehen und wie diese umgesetzt wurden.

3.1 Rest-API

Die Datenhaltung der Anwendung findet in einer nativen Anwendung auf Basis von Quarkus statt. Dieses wurde von Christoph Freimuth und Benno Steinkam im Rahmen des Moduls „Software Architektur – Konzepte und Anwendungen“ entwickelt. Diese wird über ein Rest-API Angesprochen. Die Anfragen auf die Rest-API werden mithilfe des HttpClient aus dem HttpClientModule von Angular durchgeführt. Alle Anfragen auf die Rest-API sind Account gebunden und werden über einen Cookie Authentifiziert. Der Cookie wird nach einer Anfrage auf den Endpunkt unter `/j_security_check` mit einem korrekten Passwort und Nutzernamen gesetzt. Damit der Cookie bei den Anfragen mitgesendet wird, muss die Option `withCredentials` auf `true` gesetzt werden. Für die Verschiedene Endpunkte des Rest-API existieren Services, welche die Anfragen an diesen Endpunkt umsetzen. Diese Services sind der Plan-Endpoint-Service, der Furniture-Endpoint-Service und der User-Endpoint-Service. Beispielhaft hier der Plan-Endpoint-Service:

```
import ...

@Injectable({
  providedIn: 'root'
})
export class PlanEndpointService {

  private readonly endpoint = environment.server + '/api/v1/plan';

  constructor(private http: HttpClient) { }

  public async getPlans(pageIndex: number, pageSize: number):
    Promise<Page<PlanListing>> {
    const params = new HttpParams()
      .set('page[index]', pageIndex)
      .set('page[size]', pageSize);
    return this.http.get<Page<PlanListing>>(
      this.endpoint,
      { params, withCredentials: true }
    ).toPromise();
  }

  public async getPlan(planId: number):
    Promise<PlanDetail> {...}

  public async putPlan(planId: number, plan: PlanCreateUpdate):
    Promise<PlanCreateUpdate> {...}
```

```
public async removePlan(planId: number): Promise<void> {...}  
  
public async updateFurnitureInPlan(...): Promise<void> {...}  
  
public async createFurniutureInPlan(...): Promise<void> {...}  
  
public async addPlan(plan: PlanListing): Promise<PlanListing> {...}  
}
```

3.2 Login

Um die App Nutzen zu können muss sich der Nutzer zuerst anmelden, hierfür wurde ein Login Page erstellt, die in Abbildung 2 zusehen ist. Die Anmeldung erfolgt über eine POST-Anfrage auf den „j_security_check“ Endpunkt mit Username und Passwort. Beim erfolgreichen Anmelden wird dann in der Antwort ein Cookie für die Authentifizierung mitgeschickt.

Darüber hinaus besteht die Möglichkeit sich zu registrieren, die Page ist zusehen in Abbildung 3. Um ein neuer Nutzer anzulegen, muss zuerst die Form ausgefüllt werden, als Modul haben wir dafür von Angular die „Reactive forms“ genutzt. Mit der Komponente wird verhindert, dass die Eingabe in einem falschen Format abgeschickt wird. Mit dem Drücken auf den Button „Registrieren“ wird eine POST-Abfrage an die Rest-API geschickt, wenn die Registrierung erfolgreich war, wird der Nutzer auf die Login Seite weitergeleitet, wo er sich dann anmelden kann.

register first!'." data-bbox="334 481 652 666"/>

Abbildung 2: Login Page

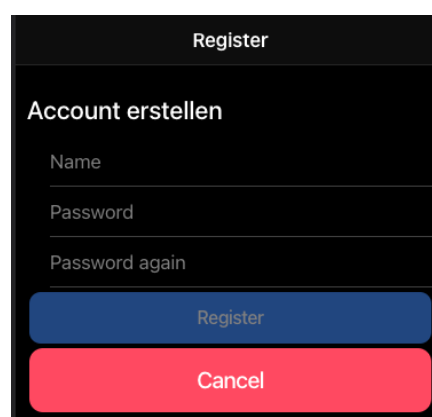


Abbildung 3: Register Page

Wenn der Nutzer sich erfolgreich eingeloggt hat, wird er an die Startseite weitergeleitet, hier ist es die Plan Verwaltung.

3.3 Plan Verwaltung

Im Folgenden wollen wir auf den ersten Tab der App eingehen, dieser dient zum Verwalten der einzelnen Raumpläne. In der unteren Abbildung ist gezeigt, wie der Tab aussieht.

Als Hauptkomponente wurde in diesem Tab eine Ionic „List“, in Kombination mit „infinite-scroll“ eingesetzt, das hat den Vorteil, dass nicht alle Elemente auf einmal geladen werden, sondern immer dann, wenn der Nutzer nach unten Scrollt. Der Infinite-scroll funktioniert so, dass immer, wenn das Ende der Liste erreicht ist, ein Event getriggert wird. Dieses Event lädt über den Endpunkt die nächsten Elemente nach und stellt sie dar.

Zudem hat der Nutzer die Möglichkeit mit dem Herunterziehen der Liste die Seite Neu zu laden. Diese Funktion wurde mit der Ionic Komponente „Refresher“ umgesetzt, dabei wird ein Event aufgerufen die alle Parameter zurücksetzt und die Seite neu lädt.

Um in den Editor zu gelangen, reicht ein Click auf den Raum und der Nutzer wird an den richtigen Raumeditor weitergeleitet.

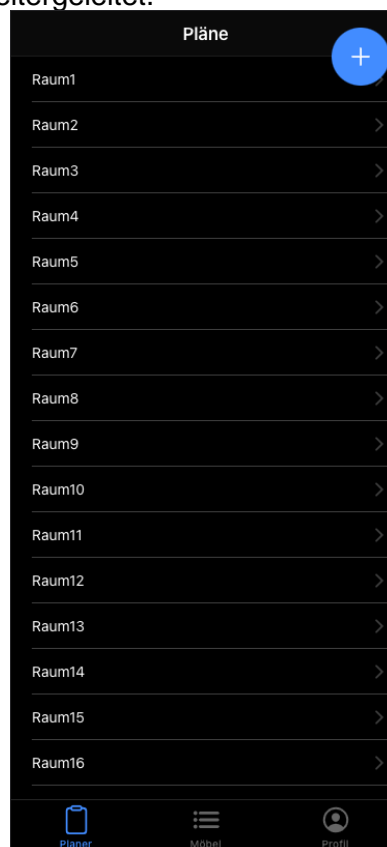
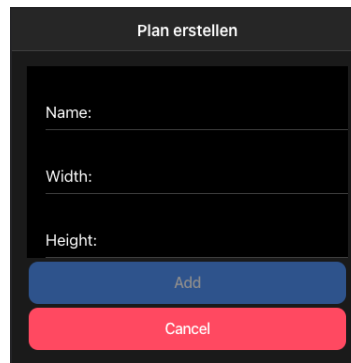


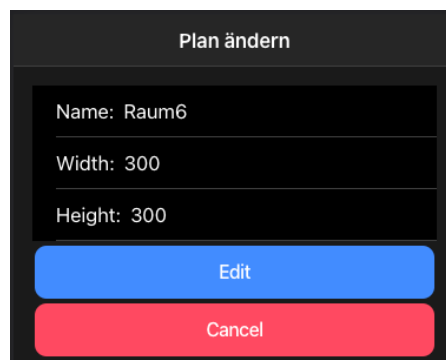
Abbildung 4: Tab Plan Verwaltung

Um einen neuen Plan anzulegen, muss der Button oben rechts betätigt werden, anschließend öffnet sich ein Modal, in dem der User wie in Abbildung 5 gezeigt dem Plan einen Namen geben und die Größe des Raumes festlegen kann. Auch hier wird das Modul „Reactive forms“ von Angular eingesetzt, um zu überprüfen, ob die Eingabe korrekt ist. Mit dem Betätigen des Add-Buttons wird über den Endpunkt der Rest-API ein neuer Raum erzeugt.



A modal dialog titled "Plan erstellen" (Create Plan) with a dark background. It contains three input fields labeled "Name:", "Width:", and "Height:". Below the fields are two buttons: a blue "Add" button and a red "Cancel" button.

Abbildung 5: Plan erstellen



A modal dialog titled "Plan ändern" (Edit Plan) with a dark background. It displays the current values for a plan: "Name: Raum6", "Width: 300", and "Height: 300". Below the fields are two buttons: a blue "Edit" button and a red "Cancel" button.

Abbildung 6: Plan ändern

Zum Ändern und Löschen eines Plans muss, das Element von rechts nach links geschoben werden. Danach ergibt sich die Möglichkeit über die zwei Button, die in Abbildung 7 gezeigt werden, das Element zu ändern oder löschen. Hierbei wurde als Komponente der Ionic „Slider“ eingesetzt, mit den Optionen Edit und Delete. Wenn der Plan bearbeitet werden soll, öffnet sich das Modal in Abbildung 6, dabei werden die alten Werte mit übergeben und angezeigt. Wenn die Werte bearbeitet wurden, werden die Änderungen über die Rest-API verarbeitet. Damit ein Plan nicht aus Versehen gelöscht wird, wurde ein „Alert“ eingebaut, der zuerst bestätigt werden muss um das Element zu Löschen.



Abbildung 7: Slider

3.4 Möbel Verwaltung

In dem Tab Möbel Verwaltung werden alle Möbel dargestellt, die ein Nutzer angelegt hat, diese Liste ist unabhängig von den Plänen und dient als Sammelbecken für alle Möbel. Über den Tab, der in der unteren Abbildung dargestellt ist, können Möbel erstellt, bearbeitet und gelöscht werden.

Die Möbel Verwaltung ist ähnlich wie die Plan Verwaltung aufgebaut, auch hier wurde eine Liste in Kombination mit infinite Scroll verwendet, darüber hinaus funktioniert das Löschen und Bearbeiten auch über einen Slider. Beim Klick auf Bearbeiten öffnet sich das Modal in Abbildung 10, dieses hat im Gegensatz zur Plan Verwaltung noch einen Color Picker, mit dem die Farbe des Möbelstücks ausgewählt werden kann.

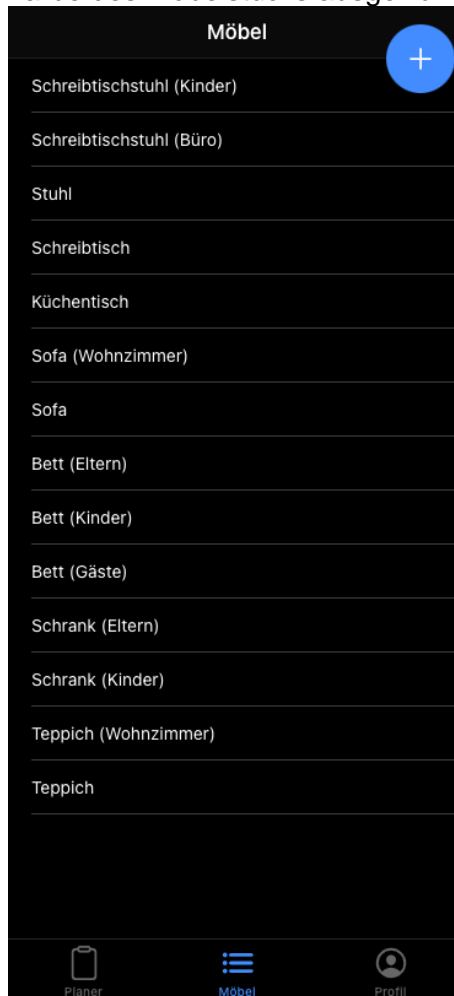


Abbildung 8: Tab Möbelverwaltung

A modal dialog titled 'Möbel ändern' (Edit Furniture) with a dark background. It contains four input fields: 'Name: Sofa', 'Width: 200', 'Height: 70', and 'Color:' with a yellow color swatch. At the bottom are two buttons: a blue 'Edit' button and a red 'Cancel' button.

Abbildung 9: Modal Möbel ändern

 A modal dialog titled 'Möbel hinzufügen' (Add Furniture) with a dark background. It contains four input fields: 'Name:', 'Width:', 'Height:', and 'Color:' with a grey color swatch. At the bottom are two buttons: a blue 'Add' button and a red 'Cancel' button.

Abbildung 10: Modal Möbel erstellen

Beim Hinzufügen von Möbeln gibt es zwei Möglichkeiten dieses zu tun, über eine manuelle Eingabe und über einen QR-Code. Die manuelle Eingabe ist dazu da, um seine eigenen Möbel hinzuzufügen und im Editor zu nutzen. Da die App für den privaten Gebrauch gedacht ist, spricht für diejenigen die einen Umzug Planen ist diese Funktion sehr wichtig, da schon vorhandene Möbel in die App eingetragen werden können. Da bei einem Umzug aber auch oft neue Möbel gekauft werden, war die Idee, Möbel auch über einen QR-Code hinzuzufügen. Diese QR-Codes könnten dann z.B. in einem Möbelhaus an den einzelnen Waren stehen. Mit der App kann dann das Möbelstück über den QR-Code hinzugefügt werden. Im Anschluss kann der Nutzer überprüfen, ob das Möbelstück in den Geplanten Raum passt oder nicht.

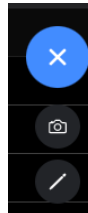


Abbildung 11: Ionic Fab-Button

Um ein Möbelstück zu erstellen, muss der Button oben Rechts, zusehen in Abbildung 8 gedrückt werden. Bei dem Button handelt es sich um ein „Floating Action Button“ von Ionic, wenn dieser gedrückt wird, erscheinen zwei weitere Buttons, zusehen in Abbildung 11.

Die manuelle Eingabe erfolgt über einen Klick auf den Stift, dabei öffnet sich das Modal in Abbildung 10, hier kann der Nutzer neben Namen und Größe des Möbelstücks auch

eine Farbe auswählen. Zur Auswahl wurde der Einfache HTML-Color Picker verwendet, die Werte werden im Anschluss über die Rest-API verarbeitet. Neben der manuellen Eingabe kann über das Kamera Symbol eine Eingabe über den QR-Code erfolgen. Das Modal für die Kamera ist in Abbildung 12 gezeigt, sobald sich das Modal geöffnet hat geht auch die Kamera an, bei dem ersten Öffnen wird nach der Berechtigung gefragt, diese sollte bestätigt werden um den QR-Code Scannen zu können. Zudem hat der Nutzer die Möglichkeit die Kameras zu wechseln, hierzu werden über ein Dropdown Menü alle Verfügbaren Kameras angezeigt. Wenn der Scann erfolgreich war, werden die Werte angezeigt und können gegebenen falls noch angepasst werden.

Abbildung 12: Modal QR-Code scanne

In Abbildung 13 ist ein Ausschnitt aus dem Code zusehen, der zeigt, wie der ZXING Scanner im Code eingebaut wurde. Der Scanner sucht beim Öffnen des Modals alle Verfügbaren Kameras und wählt die erste davon aus. Durch die Methode „onCamerasFound“ werden die Verfügbaren Kameras in dem Dropdown Menü dargestellt. Wenn der Scanner einen QR-Code erkennt, wird die Methode „onCodeResult“ getriggert und die Werte werden in die in Abbildung 12 zusehende Form geladen.

```

8 <zxing-scanner (camerasFound)="onCamerasFound($event)"
9   [formats]="['QR_CODE', 'EAN_13', 'CODE_128', 'DATA_MATRIX']"
10  (scanSuccess)="onCodeResult($event)" #scanner>

```

Abbildung 13: QR-Code Scanner

3.5 Editor

Die Editor Komponente erlaubt es einem Nutzer einen Plan zu bearbeiten. Das heißt, Möbel zu einem Plan hinzuzufügen und zu verschieben. Die Editor Komponente besteht zum größten Teil aus der main-Komponente. In der main-Komponente befindet sich ein svg-Element, welches die Arbeitsfläche des Editors darstellt. Die Einzelnen Möbelstücke und das Zimmer sind rect Elemente. Die Möbelstücke rect-Elemente werden mit der furniture-Directive versehen. In der furniture Direktive ist die Funktionalität der einzelnen Möbelstücke implementiert. Der furniture Direktive wird ein FurnitureInPlan-Objekt aus dem Plan übergeben. Dazu wird dann mithilfe des Furniture-Endpoint-Services das entsprechende Furniture-Objekt geladen. Die Ansicht wird mithilfe des transform-Attributes des svg Elementes und der rect Elemente erzeugt.

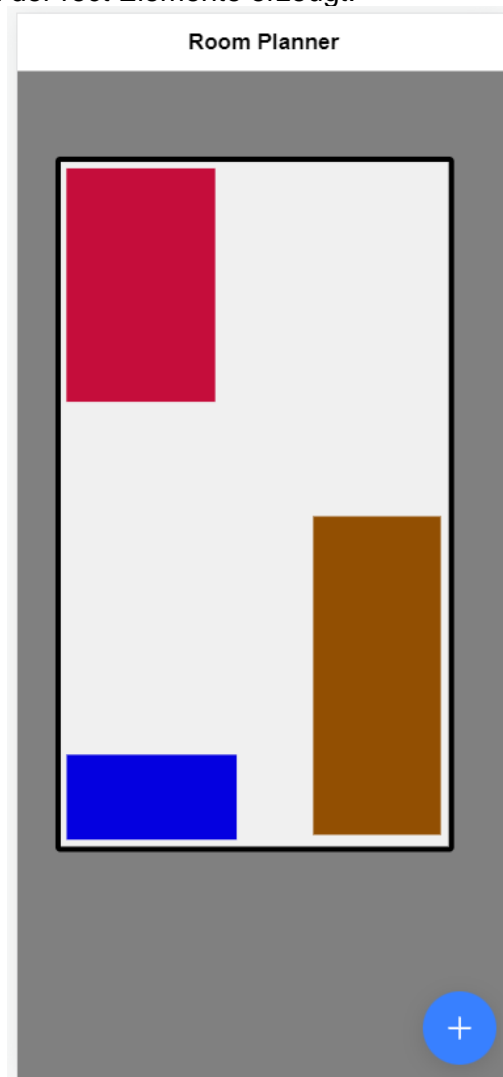


Abbildung 14 Editor

4 **Ausblick**

In diesem Kapitel wollen wir einen kurzen Ausblick geben, der zeigen soll, welche Komponenten noch integriert werden können und welches andere Einsatzgebiet es noch gibt.

4.1 **Erweiterung**

Da beim Planen eines neuen Raums oder bei einem Umzug oft mehrere Personen beteiligt sind, ist es praktisch, wenn ein Plan untereinander geteilt werden kann. Um dieses Umzusetzen, müsst aber erst die Rest-API um einige Komponenten erweitert werden, z.B. einer Freundesliste.

Wenn ein Plan schon untereinander geteilt werden kann, wäre es natürlich auch praktisch, wenn der Plan auch von mehreren Personen gleichzeitig bearbeitet werden kann. Damit dieses auch einwandfrei funktioniert muss ein Konzept entwickelt werden, dass die gemeinsame Arbeit an einem Plan regelt, damit z.B. keine Überschneidungen entstehen.

In der App ist es aktuell nur möglich Rechteckige Räume zu gestalten, auch hier sehen wir noch Potential für Verbesserung. Damit der Raum nach Belieben gestaltet werden kann muss die Rest-API Räume in unterschiedlichen Formen speichern können.

4.2 **Kommerzielle Nutzung**

Die App wurde für privat Personen entwickelt, um einen Umzug zu planen, ein weiteres Szenario, dass wir uns sehr gut vorstellen könnten, wäre der Einsatz in einem Möbelhaus. Hier wären zwei Varianten möglich, einmal könnte das Möbelhaus die App in den Onlineshop integrieren oder die App könnte denn Kunden als Planungshilfe bereitgestellt werden.

Beim Integrieren in den Onlineshop wäre der Vorteil, dass Kunden vor dem Kauf überprüfen können ob die Komponenten überhaupt in den Raum passen, da online oft nicht eingeschätzt werden kann, wie groß die Möbel wirklich sind. Desweiteren könnte eine Funktion eingebaut werden, die einen dreidimensionalen Raum generiert, die den Kunden einen Überblick über die einzelnen Möbelstücke geben soll. Wie schon vorher erwähnt lässt sich die App auch gut beim Besuch im Möbelhaus einsetzen, die Kunden können z.B., wenn ihnen ein Möbelstück gefällt dieses per QR-Code scannen und in ihren eigenen Raum einführen und nach Belieben verschieben.

Ob die App nun in einen Onlineshop integriert oder generell von einem Möbelhaus angeboten wird, muss die Präsentation der Möbel überarbeitet werden. Zum einen müssen die Möbel in Kategorien eingeteilt werden, damit der Nutzer sich bei der Masse an Möbeln zurechtfinden kann. Darüber hinaus müssen die Möbel noch besser präsentiert werden, durch Bilder und eine genauere Beschreibung der einzelnen Möbelstücke. Damit die App in einem Möbelhaus eingesetzt werden kann müssen sowohl Rest-API als auch die App um einige Komponenten erweitert werden.

5 Zusammenfassung und Fazit

Insgesamt wurde im Rahmen des Projektes eine Frontend-Webapp erzeugt, welche die Funktionen des gegebenen Systems für Endnutzer intuitiv zur Verfügung stellt. Sie ermöglicht des Weiteren eine intuitive Bearbeitung der vom System zur Verfügung gestellten Daten. Mithilfe der Ionic Bibliothek wurde für die Anwendung ein nativer Look and Feel erzeugt. Mit Capacitor wurde die Anwendung dann auch nativ für Android und IOS Geräte zur Verfügung gestellt. Die Entwicklung von Webapps mit Frameworks wie hier Angular ist insgesamt schwer zu bewerten. Positiv ist definitiv die Möglichkeit zu bewerten viele freie Bibliotheken auf Knopfdruck zur Verfügung zu haben, sowie die Erleichterte Verwaltung größerer Projekte durch die Aufteilung in Komponenten. Auch Features wie das Dynamische nachladen von Modulen erst am benötigten Moment hat viele Vorteile. Insgesamt erweitert Angular die Webentwicklung um viele Features welche in der OO-Programmierung schon lange Verwendung finden. Einer der größten Nachteile gerade auch von Ionic mit seinem erhöhten zwang von separaten Modulen für separate Seiten ist die extreme Vervielfältigung von Source-Dateien. So hat selbst ein kleines Projekt bestehend aus wenigen Seiten schnell mehr als Hundert Source-Dateien. Zwar erhöht die Aufteilung von Code auf mehrere Dateien grundsätzlich die Übersichtlichkeit, jedoch gibt es einen Punkt, an dem die Fragmentierung des Codes überhandnimmt und die Übersichtlichkeit wieder sinkt. Dieser Punkt ist unserer Meinung in Projekten welche Ionic und Angular verwenden bereits erreicht. Auch Capacitor bringt Vor- und Nachteile mit sich. Die Idee Front-End-Webapps per Webview in Nativen Anwendungen auf mobile Endgeräte zu bringen ist verlockend, da dann Webentwickler auch Mobile Anwendungen entwickeln können. In der Anwendung gibt es jedoch einige Probleme. Zunächst ist die Dokumentation für die Verschiedenen Webframeworks selten für den Support von mobilem Browser angepasst. Des Weiteren vergrößert sich der Tech Stack einer Android-App um eine Menge neuer Projekte. Wenn also ein Fehler in der Nativen Version der Webapp auftritt, ist es oft schwierig die Quelle zu finden. Infrage kommen dann nämlich: Node.js, npm, Angular, Ionic, Capacitor, Android Studio, Android-SDK und der Android Emulator. Auch mussten wir feststellen das Fehlermeldungen oft nur spärlich weitergeleitet werden. Die durch Capacitor übertragenen Anwendungen sind per Definition an die Darstellung in einer Webview gebunden und sind damit an die selben Einschränkungen wie normale Webapps gebunden. In den letzten Jahren wurde außerdem die Nutzung vieler nativer Ressourcen (Kamera, Mikrofon, Standort, etc.) für Webapps ermöglicht.

6 Literaturverzeichnis

- [1] „DENSO WAVE,“ [Online]. Available: <https://www.denso-wave.com/en/technology/vol1.html>. [Zugriff am September 2021].

