NAVIGATIONSKONZEPTE DOKUMENTATION

Saleh Chaaban & Omar Almasalmeh

31.07.2021

INHALTSVERZEICHNIS

VERWENDETE PROGRAMMIERSPRACHEN UND FRAMEWORKS

WEB-BEREICH *(SALEH CHAABAN)*

Für den Bereich Web wurde die Programmiersprache JavaScript in Kombination mit der Open-Source JavaScript Bibliothek ReactJS verwendet. Das Arbeiten mit ReactJS wurde aus dem Grund präferiert, da die Bibliothek sich gut dazu eignet, um wiederverwendbare UI-Elemente zu erstellen. ReactJS erlaubt daher das Anlegen von unabhängigen Komponenten, die sich an jeder weiteren Stelle im Quellcode mit wenigen Redundanzen und vielen Parametrisierungsmöglichkeiten wieder (unterschiedlich) nutzen lassen.

Darüber hinaus, existieren für ReactJS eine Vielzahl an nützlichen Paketen, die sich über diverse Paketmanager in ein bestimmtes Projekt oder global einbetten lassen.

Für die Implementierung genutzte Pakete sind:

- Styled-Components
- React-Router-DOM
- React-Icons

Styled-Components. Das Paket erlaubt im Wesentlichen das Designen von Komponenten direkt innerhalb der JavaScript Datei(en). Dafür definiert der Nutzer eine beliebige Komponente und gibt zusätzlich über einen SCSS ähnlichen Syntax an, wie die Komponente aussehen soll. Das Zuweisen von veränderbaren Parametern zu einzelnen Attributen stellt sich als weiterer Vorteil heraus. Komponenten können mit Hilfe dessen, dynamisch während der Laufzeit über ihre Attribute angepasst werden. Üblicherweise würden dafür für ein definiertes Event als Auslöser, eine oder mehrere CSS-Klasse(n) getauscht bzw. über die Komponente(n) gelegt werden.

React-Router-DOM. Das benannte Pakete beschäftigt sich mit der Weiterleitung auf andere Seiten bzw. genauer gesagt dem Zuordnen von URLs zu Komponenten oder Funktionen. Der vermeintlich größte Vorteil von React-Router-DOM ist die nahtlose Weiterleitung ohne einen benötigten Refresh der Seite. Die jeweilig zugeordnete Komponente oder Funktion wird, sofern die URL des Browsers mit der Route

übereinstimmt, dynamisch geladen. Somit werden Ressourcen und Zeit gespart, da nicht jedes Objekt der Seite neu gerendert werden muss, wenn es doch erhalten bleiben kann.

React-Icons. Es handelt sich dabei um eine Ansammlung frei verfügbarer und nutzbarer Icons. React-Icons schließt unter anderem bekannte Icon Bibliotheken wie die von Font Awesome oder Bootstrap ein. Als besonders vorteilhaft erweist sich der einheitliche Zugriff auf unterschiedliche Icon Bibliotheken. Viel mehr wird jedes Icon als eine eigenständige Komponente repräsentiert und ermöglicht dadurch eine vereinfachte Handhabung in React.

Zuletzt sei anzumerken, dass ReactJS für die Definition einzelner Komponenten einen an HTML angelehnten Syntax mit dem Namen JSX (= Java Script Extension) nutzt. Kleinere Unterschiede lassen sich in der Syntax ausmachen, wie am folgenden Beispiel veranschaulicht:

HTML:

<p class=‘paragraph-1‘>

JSX:

<p className=‘paragraph-1‘>

Im Gegensatz zu HTML, welches HTML-Elemente generiert, werden aus JSX, JavaScript Objekte bzw. React „Elemente“. Weitere nützliche Funktionalitäten sind etwa:

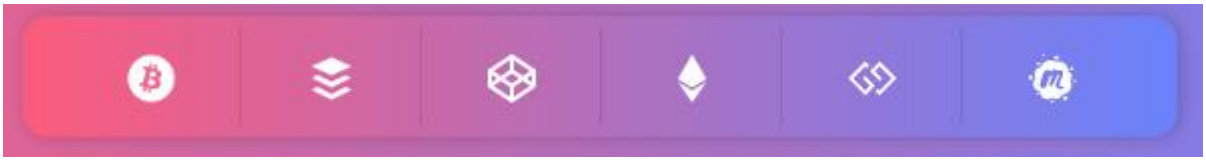
Die Möglichkeit…

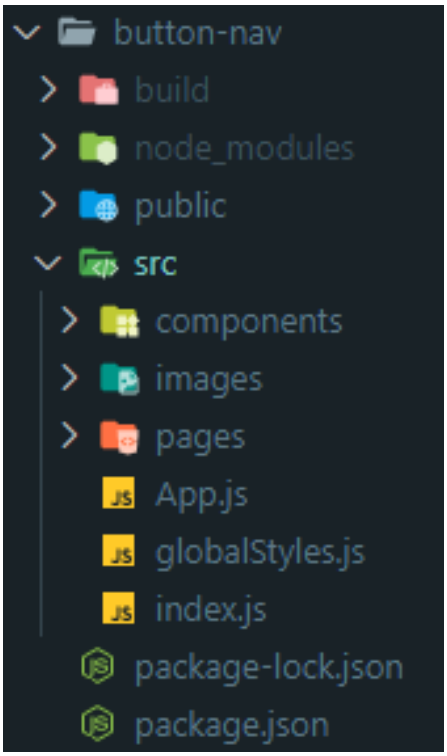
- …der Einbindung von Variablen und JavaScript Ausdrücken
- …eigene benutzerdefinierte Attribute zu erstellen
- …JSX als Rückgabetyp zu nutzen

Trotz der starken Ähnlichkeit zu HTML, ist JSX insgesamt auf Grund seiner oben genannten Eigenschaften näher mit JavaScript verwandt.

WEB-BEREICH IMPLEMENTATION *(SALEH CHAABAN)*

BUTTON-NAVIGATION





Gezeigt ist ein Ausschnitt der Ordnerstruktur für die Button-Navigation.

Public. Er enthält alle nach außen hin sichtbaren Dateien. Dabei handelt es sich noch nicht um die finale und damit Produktivversion.

Build. Der Ordner enthält alle Dateien, die hinterher veröffentlicht werden. Alle darin befindlichen Dateien sind bereits optimiert und für die Veröffentlichung bestimmt.

Src. Enthält den gesamten ReactJS Quellcode und wird darüber hinaus unterteilt in:

Components. Hier befinden sich alle wiederverwendbaren ReactJS Komponenten.

Pages. Enthält alle weiteren Unterseiten auf die hinterher verlinkt wird. Über *React Routing* wird auf diese verlinkt.

Im Folgenden wird auf alle wesentlichen Details der sich im Ordner *components* befindlichen ReactJS Komponenten eingegangen.

```
import styled from 'styled-components'
import NavContent from './NavContent'
import { Link } from 'react-router-dom'
```

Die Komponente implementiert die eigentliche Hauptnavigation. Zusätzlich werden die Inhalte der Navigation (= Icons und Verweise) in die Hauptnavigation geladen. Für den Zugriff darauf ist ein *import*-Befehl notwendig, welcher *NavContent* einbindet.

```
const Navbar = () => {
    return (
        <NavbarWrapper>
            {NavContent.map((item, index) => {
                return (
                    <>
                        <FormatedLinkIcon to={item.path} key={index}>
                            {item.icon}
                        </FormatedLinkIcon>
                        {index < '5' && <VerticalBar />}
                    </>
                )
            })}
        </NavbarWrapper>
    )
}
```

Um anschließend die Inhalte in der Navigationsleiste aufzurufen, werden über den Befehl *map* aus der Komponente *NavContent* die einzelnen Icons und Verweise an das Wickelelement *NavbarWrapper* zurückgegeben. Für die Rückgabe des jeweils passenden Elements, wird im Befehl *map* zusätzlich der Parameter *index* definiert. Dieser agiert als Schlüssel auf die Elemente und koordiniert die Zugriffe auf diese. Der Parameter *item* hingegen, ermöglicht es auf die Inhalte von NavContent zuzugreifen. So können etwa das Icon über *item.icon* oder der Pfad über *item.path* angesprochen werden.

```
{index < '5' && <VerticalBar />}
```

Für eine optische Trennung zwischen den jeweiligen Icons, wird außer für das letzte Element, zwischendrin jeweils ein schmaler vertikaler Balken eingeblendet. Der Balken selbst ist eine ansonsten funktionslose Komponente, welche über *Styled Components*

in der Datei definiert ist. Damit für das letzte Element nachstehend kein Balken eingeblendet wird, muss pro Array-Durchlauf der Index des aktuellen Elements abgefragt werden. Da es insgesamt sechs Icons gibt und die Zählung bei null beginnt, wäre der fünfte Index das letzte Element des Arrays, für das kein Balken eingeblendet werden soll.

NAVCONTENT.JS

```
const NavContent = [
    {
        icon: <FaBitcoin />,
        path: '/bitcoin',
    },
    {
        icon: <FaBuffer />,
        path: '/buffer',
    },
    {
        icon: <FaCodepen />,
        path: '/codepen',
    },
    {
        icon: <FaEthereum />,
        path: '/ethereum',
    },
    {
        icon: <FaGg />,
        path: '/gg',
    },
    {
        icon: <FaMeetup />,
        path: '/meetup',
    },
]
```

Wie bereits angemerkt, enthält die Komponente *NavContent* jegliche Inhalte, die auf der Navigationsleiste abgebildet werden. Dazu gehören unter anderem die Icons und die Pfade, welche pro Button auf jeweils eine andere Unterseite verweisen.

Im Grunde genommen ist die Komponente *NavContent* ein JavaScript Array mit sechs Einträgen und jeweils den beiden Attributen *icon* und *path* und ihren zugewiesenen Werten. Um die Navigation einfacher anpassen zu können, wurde eine Trennung zwischen Inhalt und Funktionalität vorgenommen.

```
import {
    FaBitcoin,
    FaBuffer,
    FaCodepen,
    FaEthereum,
    FaGg,
    FaMeetup,
} from 'react-icons/fa'
```

Die verwendeten Icons stammen aus der Bibliothek *Font Awesome* und werden wie im Folgenden zu sehen über das Paket *React-Icons* durch einen *import*-Befehl mit der Komponente verknüpft. Jedes Icon lässt sich im Anschluss als eigene Komponente nutzen.

In *App* werden alle definierten Komponenten als Gesamtheit zusammengebunden. Bindet eine untergeordnete Komponente jedoch bereits intern weitere Kind-Komponenten ein, so müssen diese kein weiteres Mal referenziert werden.

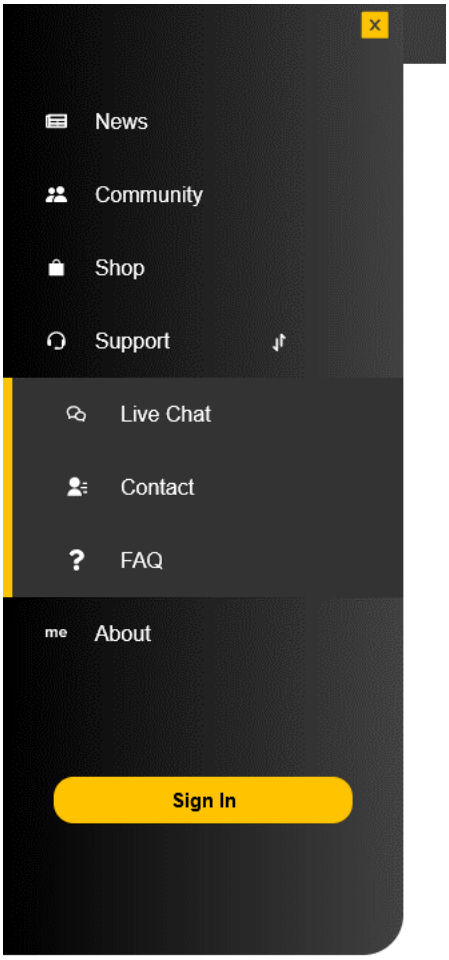
```
import { GlobalStyle } from './globalStyles'      You, seconds ago • Uncom

import { BrowserRouter as Router, Switch, Route } from 'react-router-dom'
import Navbar from './components/Navbar'

import MainContent from './pages/MainContent'
import Bitcoin from './pages/Bitcoin'
import Buffer from './pages/Buffer'
import Codepen from './pages/Codepen'
import Ethereum from './pages/Ethereum'
import Gg from './pages/Gg'
import Meetup from './pages/Meetup'
```

Neben der *Navbar*-Komponente für die Navigationsleiste werden auch alle Unterseiten über den *import*-Befehl eingebettet. Statt eines üblichen Cascading Style Sheets, wird eine globale Styling-Komponente *GlobalStyle* definiert, welche als übergreifendes allgemeines Stylesheet agiert. Weiterhin ist es notwendig, die Komponenten *BrowserRouter*, *Route* und *Switch* als Teil von React-Router-DOM, einzubinden, um die Weiterleitungsfunktionalitäten zu realisieren.

```
const App = () => {
    return (
        <Router>
            <GlobalStyle />
            <Navbar />
            <MainContent />
            <Switch>
                <Route path='/bitcoin' exact component={Bitcoin} />
                <Route path='/buffer' exact component={Buffer} />
                <Route path='/codepen' exact component={Codepen} />
                <Route path='/ethereum' exact component={Ethereum} />
                <Route path='/gg' exact component={Gg} />
                <Route path='/meetup' exact component={Meetup} />
            </Switch>
        </Router>
    )
}
```

Über eine eigene *Route*-Komponente werden zuletzt noch alle Weiterleitungen angegeben. Die Eigenschaft *exact* stellt dabei pro *Route* sicher, dass nur die Unterseite gerendert wird, welche ohne eine Abweichung in ihrer URL übereinstimmt. Jede *Route* wird innerhalb eines *Switchs* definiert, da im Falle einer Weiterleitung immer nur eine mögliche Komponente gerendert werden soll und nie mehrere gleichzeitig. Da *exact* jedoch das genannte Verhalten bereits verhindert, wäre ein *Switch* an der Stelle nicht zwingend notwendig, wird allerdings häufig empfohlen.

COLLAPSIBLE-NAVIGATION



Die Ordnerstruktur ist identisch zu jedem anderen Implementationsbeispiel, daher wird diese ab hier nicht weiter erläutert.

SIDEBAR.JS

Sidebar ist das eigentliche Wickelelement für die gesamte Navigationsleiste.

Wie bei jeder Komponente werden zu Beginn alle *imports* durchgeführt, weshalb auf diese ab hier ebenfalls nicht mehr weiter eingegangen wird.

```
const [sidebarVisible, setSidebarVisible] = useState(false)

const toggleSidebar = () => setSidebarVisible(!sidebarVisible)
```

Um festzustellen, wann die Navigationsleiste ein- bzw. ausgeklappt werden soll, wird innerhalb der *Sidebar*-Deklaration über *useState* ein Zustand mit dem initialen Wert *false* (= Navigation eingeklappt) erzeugt. Der Zustand lässt sich bei einem Aufruf der Variable *toggleSidebar* in den jeweilig gegenteiligen überführen.

```
<StyledMainNav>
    <StyledOpenIcon onClick={toggleSidebar} />
</StyledMainNav>
<StyledSidebar isVisible={sidebarVisible}>
    <IconWrapper>
        <StyledCloseIcon onClick={toggleSidebar} />
    </IconWrapper>
```

Ausgelöst wird das Öffnen und Schließen der Navigationsleiste dann über das Hamburger-Symbol *StyledOpenIcon* bzw. das X-Symbol *StyledClosedIcon*, welche über das Attribut *onClick* mit *toggleSidebar* den Zustand der Seitenleiste verändern.

```
{Data.map((item, index) => {
    return <SidebarItem item={item} index={index} />
})}
```

Die übergeordneten Einträge werden dann wie im Navigationsbeispiel zuvor aus dem Array *Data* extrahiert, über die *map*-Funktion jeweils in eine Komponente *SidebarItem* gewickelt und auf die Seitenleiste gebracht.

SIDEBARITEM.JS

```
const SidebarItem = ({ item, index }) => {
    const [subNavIsActive, setSubNavIsActive] = useState(false)

    const toggleSubNav = () => setSubNavIsActive(!subNavIsActive)
```

Um die jeweiligen Elemente des Arrays passend und in der richtigen Reihenfolge abzubilden, werden der Index und das Objekt in *SidebarItem* als Parameter erwartet.

Zusätzlich wird, wie zuvor ein Zustand definiert, welcher dazu dient, das Submenü für den Eintrag „Support“ zu öffnen und wieder zu schließen. Der Zustand lässt sich wiederum über die Variable *toggleSubNav* verändern.

```
<StyledLink
    key={index}
    to={item.path}
    onClick={item.subMenu && toggleSubNav}>
    {item.icon}
    <StyledTitle>{item.text}</StyledTitle>
    <IconWrapper subNavActive={subNavIsActive}>
        {item.rightIcon}
    </IconWrapper>
</StyledLink>
```

Durch einen Klick auf den Eintrag „Support“, welcher das Submenü implementiert, wird über das Attribut *onClick* wieder der Ein- und Ausfahrmechanismus ausgelöst.

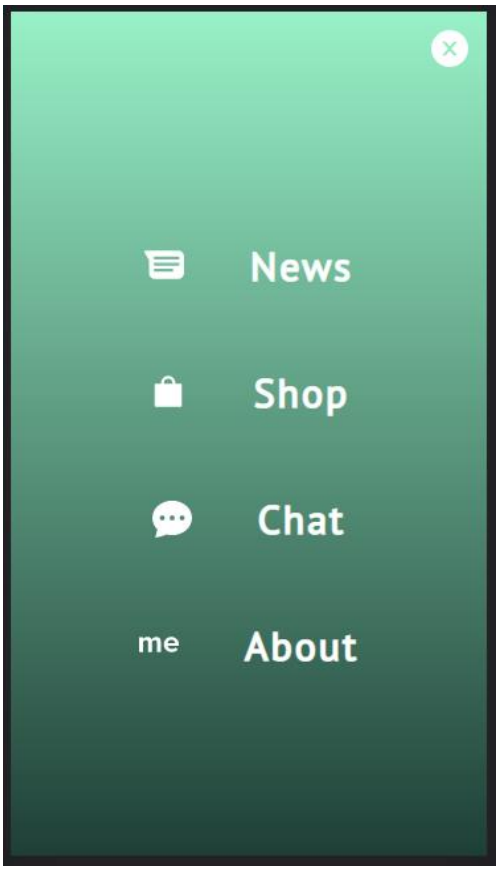
```
{subNavIsActive &&
    item.subMenu.map((item, index) => {
        return <DropdownItem item={item} index={index} />
    })}
```

Anschließend werden die Einträge für das Submenü aus dem Sub-Array *subMenu* im übergeordneten Array *Data* extrahiert, als Komponente *DropdownItem* verpackt und unterhalb der Sektion „Support“ abgebildet.

DROPDOWNITEM.JS

```
const DropdownItem = ({ item, index }) => {
    return (
        <StyledDropdownLink key={index} to={item.path}>
            {item.icon}
            <StyledTitle>{item.text}</StyledTitle>
        </StyledDropdownLink>
    )
}
```

Auch an der Stelle werden wieder das Objekt und der Index des Subeintrags als Parameter in *DropdownItem* erwartet, um darauf die passenden Inhalte und Verlinkungen darzustellen.



MENU.JS

Wie zuvor wird ein *useState* für das Ein- und Ausfahren der Navigation definiert. Über das Attribut *onClick* wird der Zustand der Navigation wiederum geändert.

Auch hier wird die *map*-Funktion angewandt, um die Elemente aus dem Array in *MenuContent.js* in klickbare Einträge umzuwandeln. Genauer gesagt werden die Elemente in *Button*-Komponenten gewickelt und anschließend mit ihrem jeweiligen Icon und Text im Menü innerhalb des Elements *ItemWrapper* dargestellt.

```
const Menu = () => {
    const [menuVisible, setMenuVisible] = useState(false)

    const toggleMenu = () => setMenuVisible(!menuVisible)

    return (
        <>
            <StyledFaBars onClick={toggleMenu} isvisible={menuVisible} />
            <MenuWrapper isvisible={menuVisible}>
                <StyledGrClose onClick={toggleMenu} isvisible={menuVisible} />
                <ItemWrapper>
                    {MenuContent.map((item, index) => {
                        return (
                            <Button
                                path={item.path}
                                key={index}
                                onClick={toggleMenu}>
                                <IconWrapper>{item.icon}</IconWrapper>
                                <TextWrapper>{item.text}</TextWrapper>
                            </Button>
                        )
                    })}
                </ItemWrapper>
            </MenuWrapper>
        </>
    )
}
```

MEGA-MENU-NAVIGATION

TOPNAV.JS

```
const TopNav = () => {
    return (
        <Navbar>
            <ContentWrapper>
                <LogoWrapper as='li'>
                    <AiFillTaobaoCircle size='45px' />
                    Mega Menu
                </LogoWrapper>
                <NavItems />
            </ContentWrapper>
        </Navbar>
    )
}
```

TopNav bildet das Wickelelement für die Navigationsleiste, welche neben dem Logo alle klickbaren Felder enthält.

```
const [subnavActive, setSubnavActive] = useState(false)
const [subnavActive2, setSubnavActive2] = useState(false)

const enableSubnav = () => setSubnavActive(true)
const disableSubnav = () => setSubnavActive(false)

const enableSubnav2 = () => setSubnavActive2(true)
const disableSubnav2 = () => setSubnavActive2(false)
```

Da für die Einträge „Projects“ und „Ideas“ der Navigationsleiste jeweils eine Mega Menü Ansicht implementiert ist, werden in diesem Fall zwei *useStates* definiert. Für beide existiert jeweils eine einzelne Funktion zum Ein- bzw. Ausblenden der jeweiligen Navigationsleiste, da eine versuchte Zusammenführung zu einer Funktion zu einem Fehlverhalten der Navigation führte.

```
<Item
    align='1'
    onMouseOver={enableSubnav}
    onMouseLeave={disableSubnav}>
    <IconWrapper className='animate_icon'>
        <AiFillExperiment />
    </IconWrapper>
    <TitleWrapper className='animate_title'>Projects</TitleWrapper>
    <Subnav active={subnavActive} />
</Item>
<Item
    align='2'
    onMouseOver={enableSubnav2}
    onMouseLeave={disableSubnav2}>
    <IconWrapper className='animate_icon'>
        <AiFillBulb />
    </IconWrapper>
    <TitleWrapper className='animate_title'>Ideas</TitleWrapper>
    <Subnav2 active={subnavActive2} />
</Item>
```

Das jeweilige Mega Menü wird mit Hilfe des Attributs *onMouseOver*, also einem schweben mit der Maus über den Eintrag aktiviert und durch ein Verlassen des Mega Menüs mit der Maus mit Hilfe des Attributs *onMouseLeave* wieder deaktiviert. Über das definierte Attribut *active* wird den einzelnen Menüs jeweils mitgeteilt, ob sie angezeigt werden sollen.

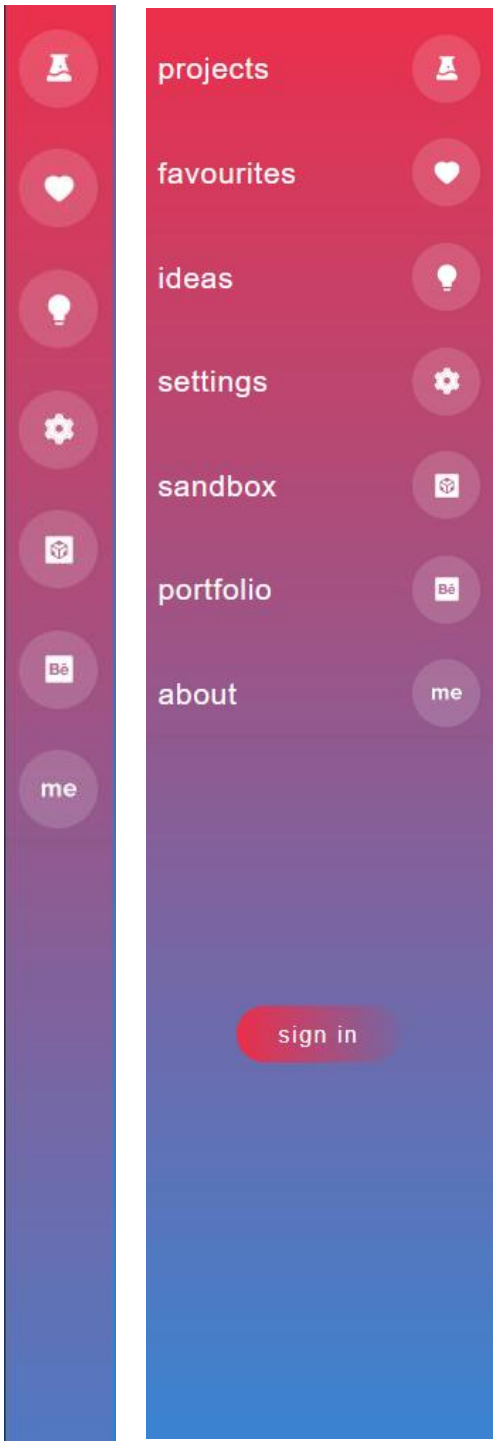
```
visibility: ${({ active }) => (active ? 'visible' : 'hidden')};
opacity: ${({ active }) => (active ? '100%' : '0')};
```

Nachdem der genannte Parameter *active* von *Subnav* und *Subnav2* entgegengenommen werden, wird dieser innerhalb des Stylesheets für die Komponente ausgewertet und verändert damit abhängig vom Variablenzustand seine Transparenz und Sichtbarkeit.

```
const Subnav = ({ active }) => {
    return (
        <SubnavWrapper active={active}>
            {NavContent.map((subItem, subIndex) => {
                return (
                    <Item
                        to={subItem.path}
                        key={subIndex}
                        picURL={subItem.pictureURL}>
                        <FormatedCaption>{subItem.pictureDesc}</FormatedCaption>
                    </Item>
                )
            })}
        </SubnavWrapper>
    )
}
```

Über die jeweiligen *map*-Befehle, welche über die Elemente des Arrays in *NavContent* bzw. *NavContent2* laufen, werden zum Schluss alle Einträge mit ihrem zugehörigen Bild, Pfad und ihrer Beschreibung innerhalb der Wickelkomponente *SubnavWrapper* als *Item*-Element im Mega Menü abgebildet.

OFF-CANVAS-NAVIGATION



OFFCANVAS.JS

```
const OffCanvas = ({ visible, handle }) => {
    return (
        <OffCanvasMenu
            canvasVisible={visible}
            onMouseOut={() => handle()}
            onMouseOver={() => handle()}>
            <ContentWrapper>
                {MenuContent.map((item, index) => {
                    return <MenuItem key={index} item={item} />
                })}
            </ContentWrapper>
            <ButtonWrapper>
                <Button />
            </ButtonWrapper>
        </OffCanvasMenu>
    )
}
```

Dem Off-Canvas-Menü wird über den Parameter *visible* mitgeteilt, ob die Komponente ausgefahren werden soll. Zusätzlich bekommt sie den Parameter *handle* übergeben, welcher eine Funktion zum Ändern des Sichtbarkeitsparameters in *App.js* ist. Die Funktion wird ausgelöst, sobald sich der Mauszeiger über der Off-Canvas-Seitenleiste befindet (= *onMouseOver*) bzw. sobald dieser ihn wieder verlässt (= *onMouseOut*).

Dann wiederum iteriert die Funktion *map* über das Array *MenuContent* und bildet die Menüeinträge im *ContentWrapper*-Element als *MenuItem*-Komponenten ab.

MENUITEM.JS

```
const MenuItem = ({ index, item }) => {
    return (
        <Item to={item.path} key={index}>
            {item.text}
            <IconWrapper>{item.icon}</IconWrapper>
        </Item>
    )
}
```

Ein *MenuItem* bekommt pro Menüeintrag jeweils wieder einen Index und das Array-Objekt übergeben und legt pro Element den Pfad, den Text und das Icon fest.

APP.JS

```
const [canvasVisible, setCanvasVisible] = useState(false)
const toggleCanvas = () => setCanvasVisible(!canvasVisible)
```

```
<OffCanvas handle={toggleCanvas} visible={canvasVisible} />
```

Die Zustandsvariable *canvasVisible* und die Funktion *toggleCanvas* zur Änderung der Zustandsvariable werden wie bereits erläutert an der Stelle zunächst an die Komponente Off-Canvas übergeben.

```
<Switch>
    <Route
        path='/projects'
        exact
        render={(props) => (
            <Projects {...props} shifted={canvasVisible} />
        )}
    />
```

Um beim Ausklappen der Off-Canvas-Seitenleiste die jeweilig aktive Seite ein Stück nach rechts zu verschieben, ist eine Benachrichtigung dieser Komponente entscheidend. Sie sollte jedes Mal benachrichtigt werden, sobald die Seitenleiste ihren Zustand ändert. Dafür werden zunächst wie üblich über einen Switch und mehrere *Routes* die jeweiligen Unterseiten zunächst verlinkt. Der Unterschied hier besteht dann aber darin, dass über das Attribut *render* im Vergleich zum üblichen *component*-

Attribut auch zusätzliche Eigenschaften an die Komponente übergeben werden können. So gelingt es, dass über das eigens definierte Attribut *visible* der Zustand der Seitenleiste mitgeteilt wird.

```
const About = ({ shifted }) => {
    return (
        <ContentWrapper shifted={shifted}>
```

```
left: ${({ shifted }) =>
    shifted ? 'calc(15.5rem + (15.5rem - 13rem))' : '0'};
```

Anschließend wird der Zustand von der jeweiligen Unterseite erfasst und im globalen Stylesheet des Elements *ContentWrapper* ausgewertet. Dabei wird, falls der Wert dafür auf *true* gesetzt ist, der Seiteninhalt um eine bestimmte Länge nach rechts verschoben.

SLIDER-NAVIGATION

Bei der vorliegenden Implementation handelt es sich um ein Bilderkarussel. Jedes Bild verfügt dabei zusätzlich über einen Titel und eine Beschreibung. Durch einen Klick auf eines der Kästchen im statischen Navigationsmenü wird zum jeweils passenden Bild gewechselt. Die Implementation soll zeigen, dass Navigation nicht nur bedeutet auf andere Seiten weiterzuleiten, sondern, dass auch eine Navigation innerhalb von einzelnen Komponenten auf einer Seite existieren kann.

SLIDER.JS

```
const [shiftUp, setShiftUpBy] = useState('0')
```

Zu Beginn der Implementation der Komponente *Slider* wird ein *useState* definiert, welcher die aktuelle Position des Bildes innerhalb des Bilderkarussels widerspiegelt.

```
<SlidebarWrapper>
    <Radio
        name='currentImage'
        id='img1'
        onClick={() ⇒ setShiftUpBy('0')}
    />
    <label for='img1'></label>
    <Radio
        name='currentImage'
        id='img2'
        onClick={() ⇒ setShiftUpBy('-20%')}
    />
    <label for='img2'></label>
    <Radio
        name='currentImage'
        id='img3'
        onClick={() ⇒ setShiftUpBy('-40%')}
    />
    <label for='img3'></label>
    <Radio
        name='currentImage'
        id='img4'
        onClick={() ⇒ setShiftUpBy('-60%')}
    />
    <label for='img4'></label>
    <Radio
        name='currentImage'
        id='img5'
        onClick={() ⇒ setShiftUpBy('-80%')}
    />
    <label for='img5'></label>
</SlidebarWrapper>
<SlidingImages shift={shiftUp} />
```

Dabei verändert jeder Button der statischen Navigationsleiste mit Hilfe des Setters *setShiftUpBy* innerhalb des *onClick*-Attributs die aktuelle Bildposition hin zu seinem zugehörigen Bild. Dabei reflektieren der Wert 0% das erste und der Wert 80% das letzte Bild.

Die entsprechenden Bilder sind bereits von Beginn an alle in vertikaler Reihenfolge innerhalb des Wickelelements *SlidebarWrapper* angeordnet, wobei der Überlauf verhindert wird, so dass augenblicklich nur das jeweilige Bild zu erkennen ist, welches sich gerade innerhalb des Bilderrahmens befindet.

SLIDINGIMAGES.JS

```
const SlidingImages = ({ shift }) ⇒ {
    return (
        <SlideWrapper shift={shift}>
            {ImageURLs.map((item, index) ⇒ {
                return (
                    <ImageWrapper key={index} url={item.url}>
                        <ContentWrapper>
                            <StyledTitle>{item.title}</StyledTitle>
                            <StyledText>{item.desc}</StyledText>
                        </ContentWrapper>
                    </ImageWrapper>
                )
            })}
        </SlideWrapper>
    )
}
```

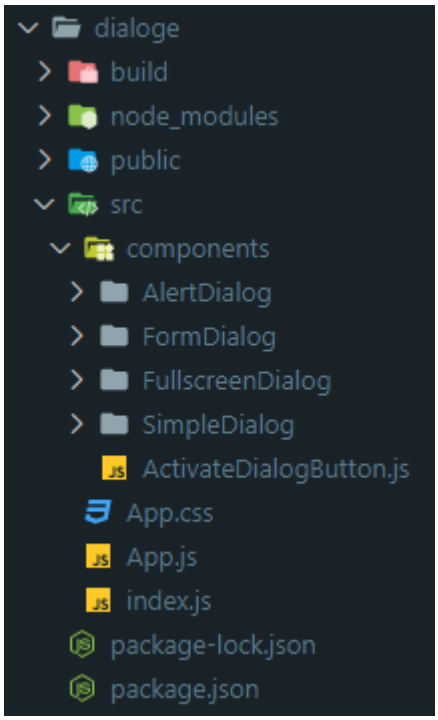
```
transform: translateY(${(props) ⇒ props.shift});
```

Die Bildersammlung wird durch die Komponente *SlidingImages* dargestellt und erwartet als Parameter den aktuellen Versatz (= Position) des Bildes. Der Versatz wird

anschließend dem Element *SlideWrapper* mitgeteilt, welcher eine Auswertung des mitgelieferten Arguments vornimmt und innerhalb seines Stylesheets eine Anpassung der Position auf der Y-Achse des Wickelelements vornimmt.

Jedes Bild wird im Anschluss über die *map*-Funktion durch ihre jeweilige URL aus dem Array in *ImageURLs* als *ImageWrapper*-Element im Bilderkarussel abgebildet. Zusätzlich zur URL des Bildes enthält das Array auch Informationen über den Titel des Bildes, die Bildbeschreibung und Alt-Beschreibung.

DIALOGE



Insgesamt wurden vier Beispieldialoge implementiert, welche jeweils ihren eigenen Ordner innerhalb von *components* besitzen. Jeder Dialog wird dabei über einen Button ausgelöst, welcher als Komponente *ActivateDialogButton* definiert ist von der Komponente *App* aufgerufen wird.

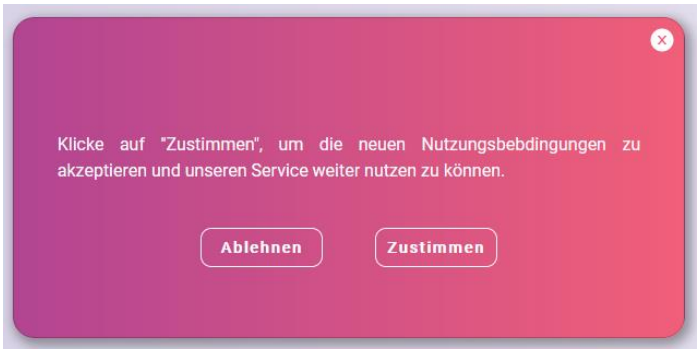
```
const [currentDialog, setCurrentDialog] = useState('none')
```

Zu Beginn wird ein *useState* definiert, der darüber bestimmt, welcher Dialog aktiviert werden soll. Die Zustandsvariable *currentDialog* enthält dabei zu jedem Zeitpunkt einen eindeutig zu einem Dialog zuordenbaren String. Initial ist der Wert „none“ festgelegt, es soll also zu Beginn kein Dialog angezeigt werden.

```
<ContentWrapper>
    <ActivateDialogButton
        onClick={() => setCurrentDialog('alert')}
        dialogOpen={currentDialog !== 'none' ? true : false}>
        Slide In Alert Dialog
    </ActivateDialogButton>
    <ActivateDialogButton
        onClick={() => setCurrentDialog('full')}
        dialogOpen={currentDialog !== 'none' ? true : false}>
        Full Screen Dialog
    </ActivateDialogButton>
```

Anhand der beiden Beispiele lässt sich wie beschrieben ihr eindeutiger Bezeichner erkennen, welcher durch einen Klick auf den zugehörigen Button mit *setCurrentDialog* eine Zustandsänderung bewirkt und den passenden Dialog dazu öffnet.

Im Folgenden werden in kürze die wichtigsten Codestellen für die beiden Beispieldialoge *AlertDialog* und *FullscreenDialog* erläutert.



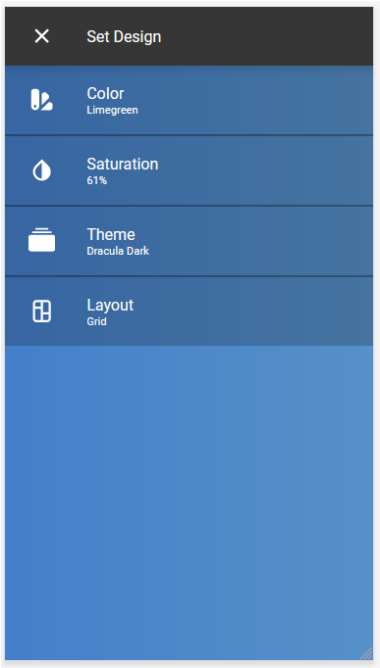

```
opacity: ${({ displayDialog }) => (displayDialog ? '100%' : '0%')};
left: ${({ displayDialog }) => (displayDialog ? '0' : '-100%')};
```

Über das Attribut *displayDialog* teilt die Komponente ihrem Stylesheet mit, dass sich der Dialog öffnen bzw. schließen soll.

```
const AlertDialog = (props) => {
    return (
        <StyledDialog displayDialog={props.dialogOpen}>
            <CloseButton onClick={() => props.handle('none')} />
            <TextContainer>{props.children}</TextContainer>
            <ButtonContainer>
                <ConfirmButton onClick={() => props.handle('none')}>
                    Ablehnen
                </ConfirmButton>
                <ConfirmButton onClick={() => props.handle('none')}>
                    Zustimmen
                </ConfirmButton>
            </ButtonContainer>
        </StyledDialog>
    )
}
```

Durch einen Klick auf einen der beiden dargestellten Buttons wird die Zustandsvariable *currentDialog* in der Komponente *App* auf „none“ gesetzt, wodurch sich der Dialog wieder schließt.

FULLDIALOG.JS




```
const FullDialog = (props) => {
    return (
        <StyledDialog displayDialog={props.dialogOpen}>
            <DialogBar>
                <StyledRiCloseFill onClick={() => props.handle('none')} />
                Set Design
            </DialogBar>
```

Analog zum vorherigen Dialog wird in diesem Codeabschnitt das Stylesheet der Komponente über das Attribut *displayDialog* benachrichtigt. Durch einen Klick auf das X-Symbol wird die Zustandsvariable wiederum auf „none“ gesetzt, wodurch der Dialog geschlossen wird.

WEITERE DIALOGE

Alle weiteren Dialoge funktionieren im Kern identisch zu den beiden veranschaulichten Beispielen und werden daher nicht weiter erläutert.

DESKTOP-BEREICH IMPLEMENTATION *(OMAR ALMASALMEH)*

INTRO

Für den Bereich Desktop wurde die *Windows Presentation Foundation (WPF)* mit der Programmiersprache *C#* und *XAML (Extensible Application Markup Language)* verwendet. Zusätzlich wurde die Open-Source *Material Design UI* Bibliothek verwendet.

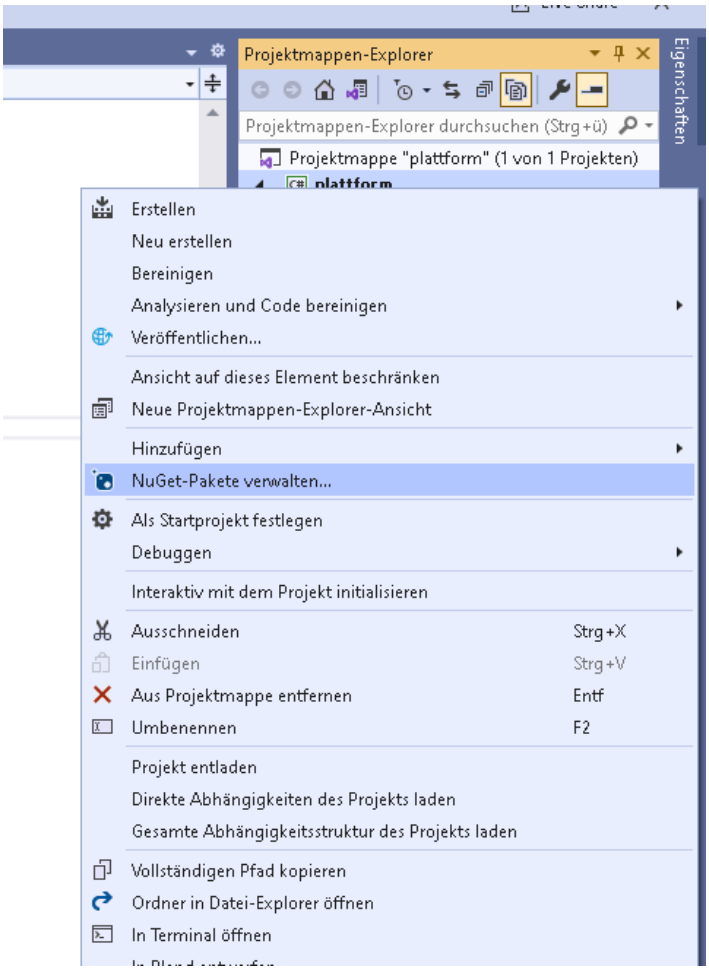
Material Design ist eine von Google Designsprache. Sie verbindet die Gestaltungsregeln des klassischen Grafik-Designs mit den Möglichkeiten digitaler Benutzeroberflächen.

INSTALLATION & AUSFÜHRUNG:

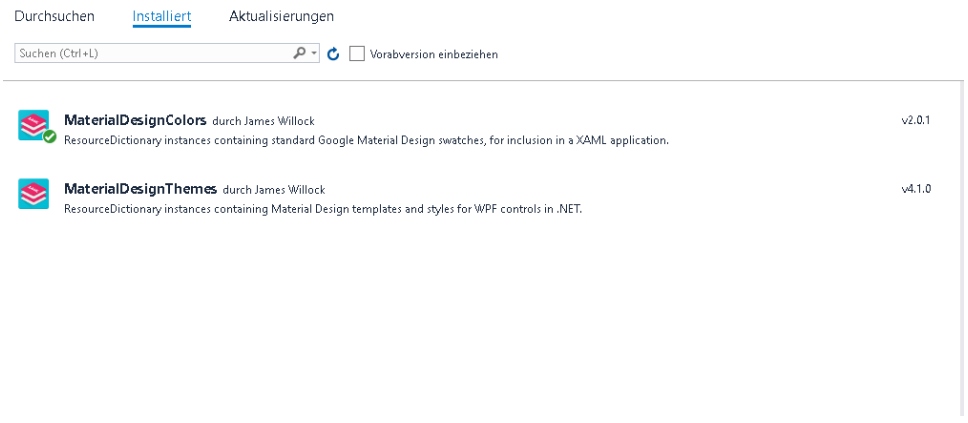
1. Installieren Sie zuerst Visual Studio Installer.
2. Folgen Sie die Schritte im Link: https://docs.microsoft.com/en-us/visualstudio/install/install-visual-studio?view=vs-2019
3. Wählen Sie bei Workloads .NET-Desktopentwicklung aus.
4. Clonen Sie das Projekt vom Github auf Ihrem Computer.
5. Öffnen Sie Visual Studio.
6. Wählen Sie “Projekt oder Projektmappe” öffnen aus.
7. Wählen Sie das Projekt aus (ZB. Plattform.sln).

INSTALLATION MATERIAL DESIGN UI

1. Im Editor öffnen Sie Projektmappen-Explorer (wenn es bereits nicht geöffnet, gehen Sie oben auf Ansicht und öffnen sie den Projektmappen-Explorer).
2. Klicken Sie mit der rechten Maustaste auf dem Projektname und wählen Sie NuGet-Pakete verwalten.

3. 

4. Stellen Sie sicher, dass die Pakete “MaterialDesignThemes” und “MaterialDesignColors” schon installiert sind (wenn nicht, installieren Sie sie).



5.
6. Die App ist ausführbar.

PROJEKT „PLATTFORM“

Das Projekt “Plattform” ist eine Musik-App UI. Es besteht aus einem Anmelden, Registrieren-Seite und Hauptseite. In der Anmelden Seite kann man E-Mail und

Password eingeben. Es gibt auch die Möglichkeit beim Password-vergessen das Password zurückzusetzen. In der Registrieren Seite kann man Namen, E-Mail, Password, Geburtsdatum eingeben. Die Hauptseite besteht aus einem Navigationsleiste, Suche Bar, Profiletaste, Benachrichtigungstaste und Inhalt der Seite. Man kann zwischen verschiedene Seiten navigieren. In der Startseite kann man Albums sehen. In den Kategorien kann man beim Klicken auf einer Kategorie zu der Songs-liste kommen und entsprechend zurücknavigieren. Man kann im Suche-Bar etwas eingeben und zu einem Suchergebnis kommen. Die Profiletaste und Benachrichtigung öffnen mit ein Dropdown Menü.

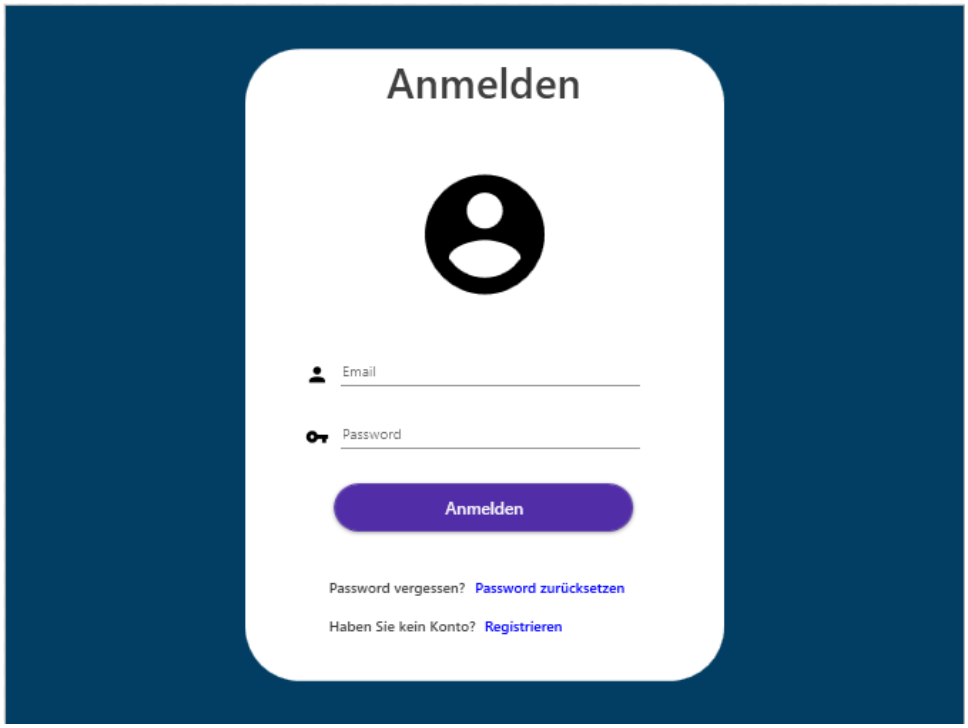
FOKUS

Der Fokus im Projekt *“plattform”* liegt bei der Gestaltung der UI und bei den eingesetzten Dialogen, Menüs und Navigationsrichtungen.

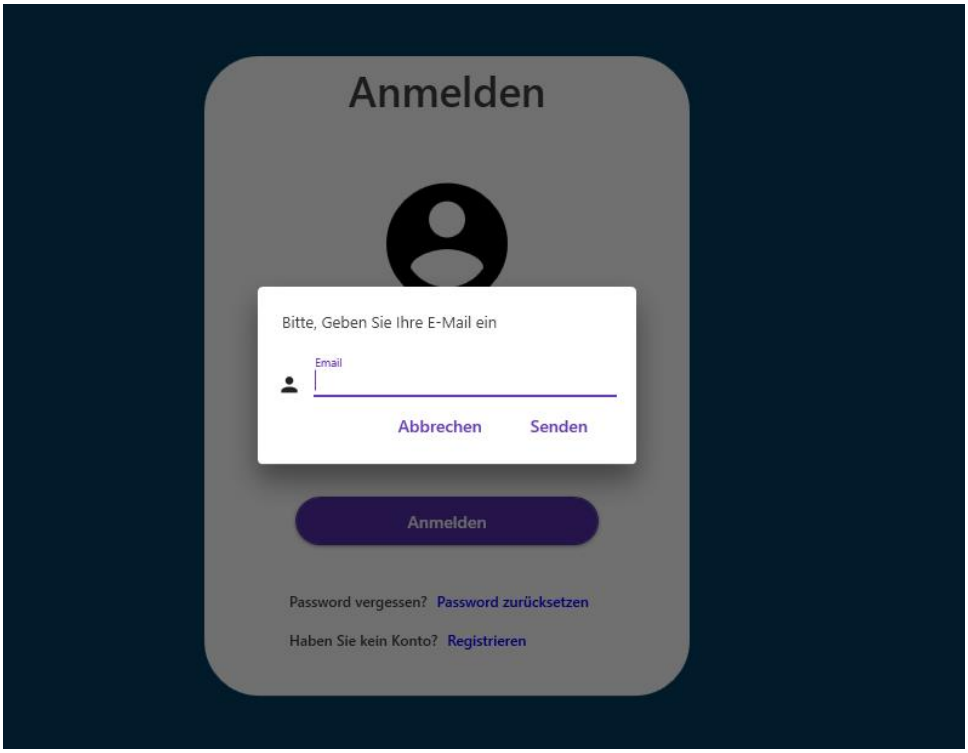
IMPLEMENTIERUNG & BESCHREIBUNG

ANMELDEN SEITE

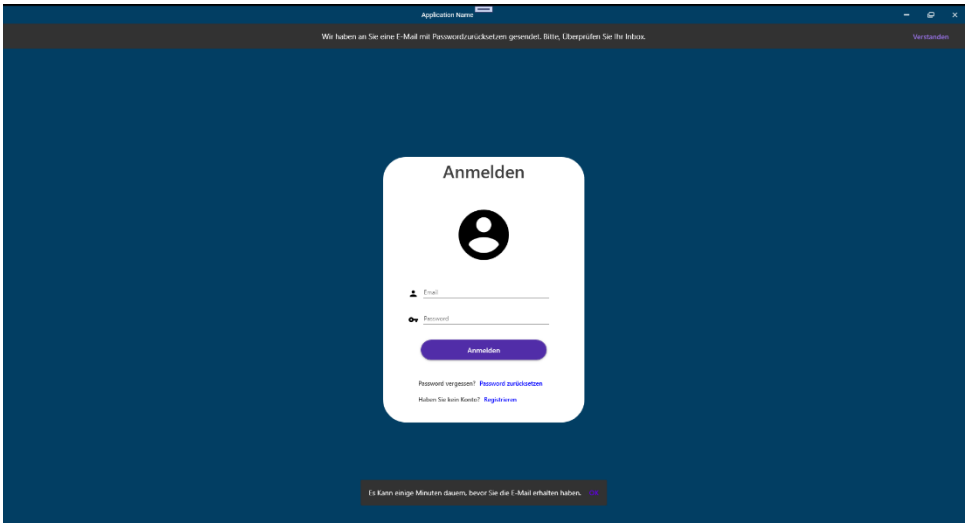
Es besteht aus zwei Textboxes mit entsprechend Icons und eine Anmeldentaste, Password zurücksetzen-, Registrieren-Taste.



Mit Password zurücksetzen öffnet sich ein Bestätigungsdialog

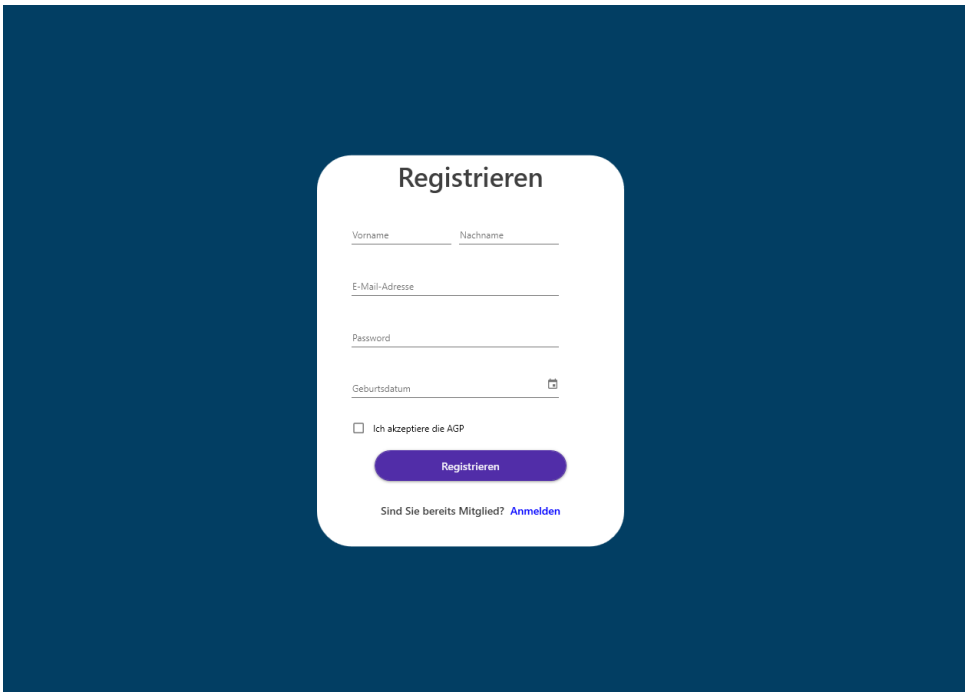


Wenn man auf Senden klickt, öffnet sich ein Banner und eine Reihe von Snackbar-Nachrichten. Die Reihe von Snackbarnachrichten kann man mit einem einzelnen Klick auf Ok-Taste stoppen. Und der Banner schließt, wenn man auf “verstanden” klickt.

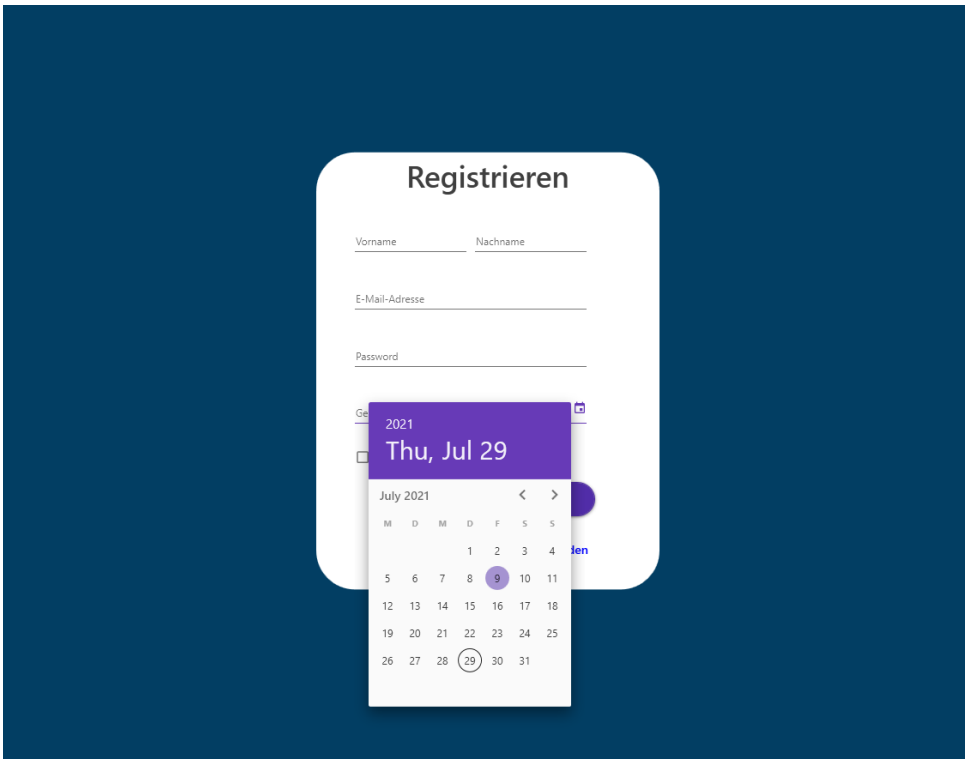


REGISTRIEREN SEITE

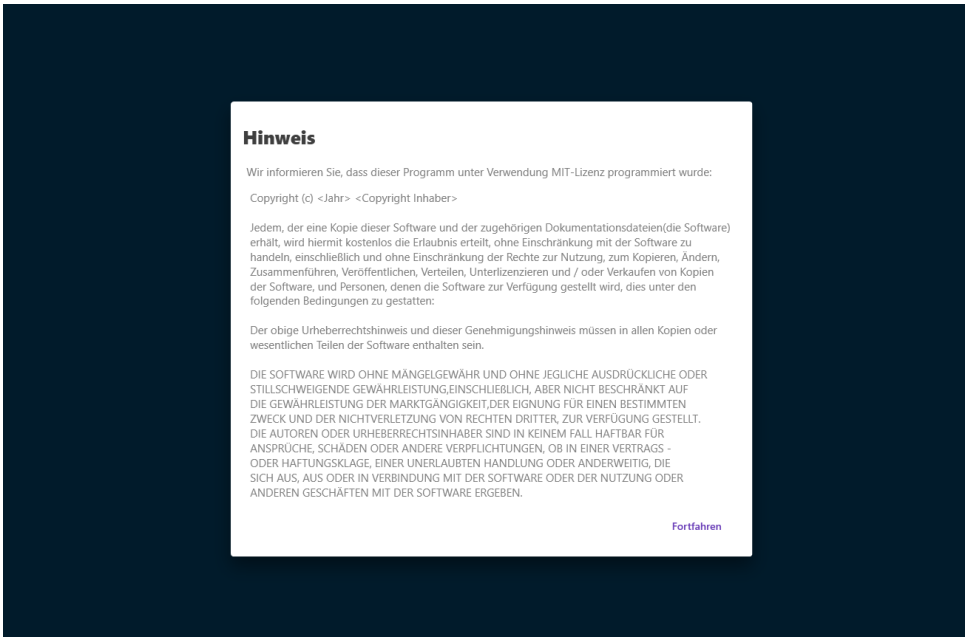
In der Registrieren Seite kann man Namen, E-Mail, Password, Geburtsdatum eingeben.



Ein Klick auf das Datum öffnet ein Bestätigungsdialog



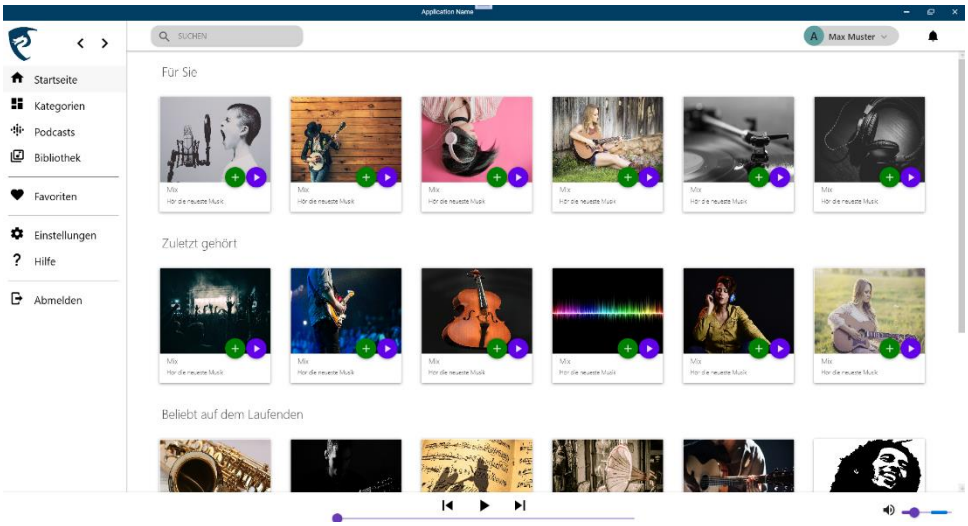
Ein Klick auf Registrieren öffnet ein Dialog mit einer einzelnen Aktion.



DIE HAUPTSEITE

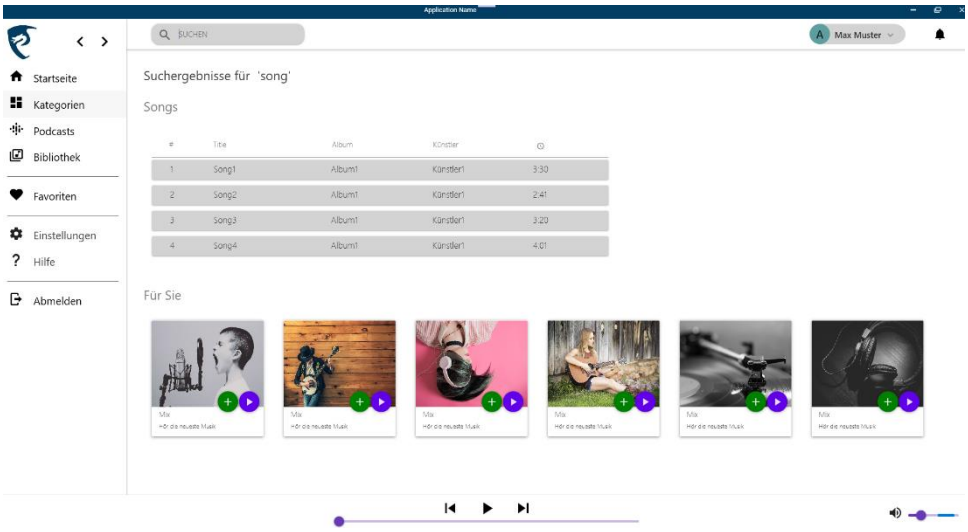
Man kann durch verschiedene Seiten navigieren.

Die Startseite enthält Albums.

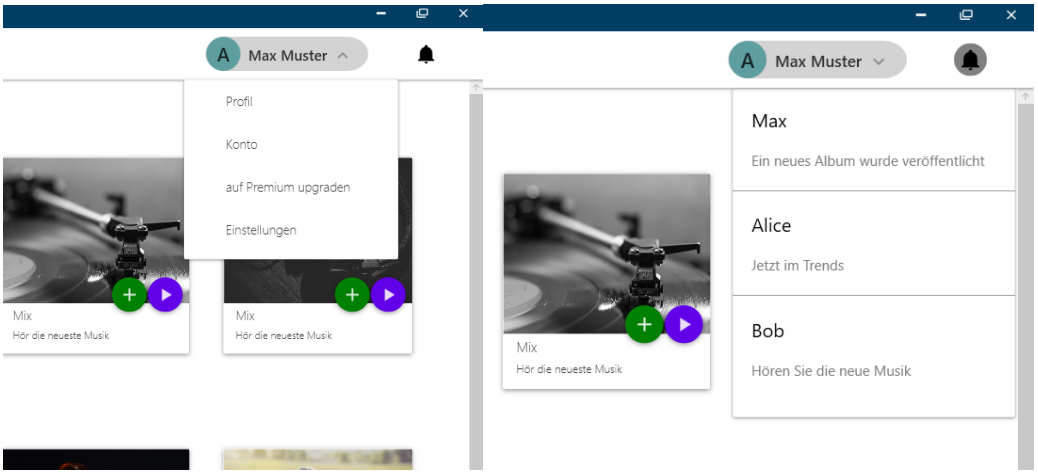


Die Kategorie enthält verschiedene Songs-Kategorien mit einem Klick auf eine Kategorie kommt man zu einer Songs-liste und kann entsprechend zurücknavigieren.

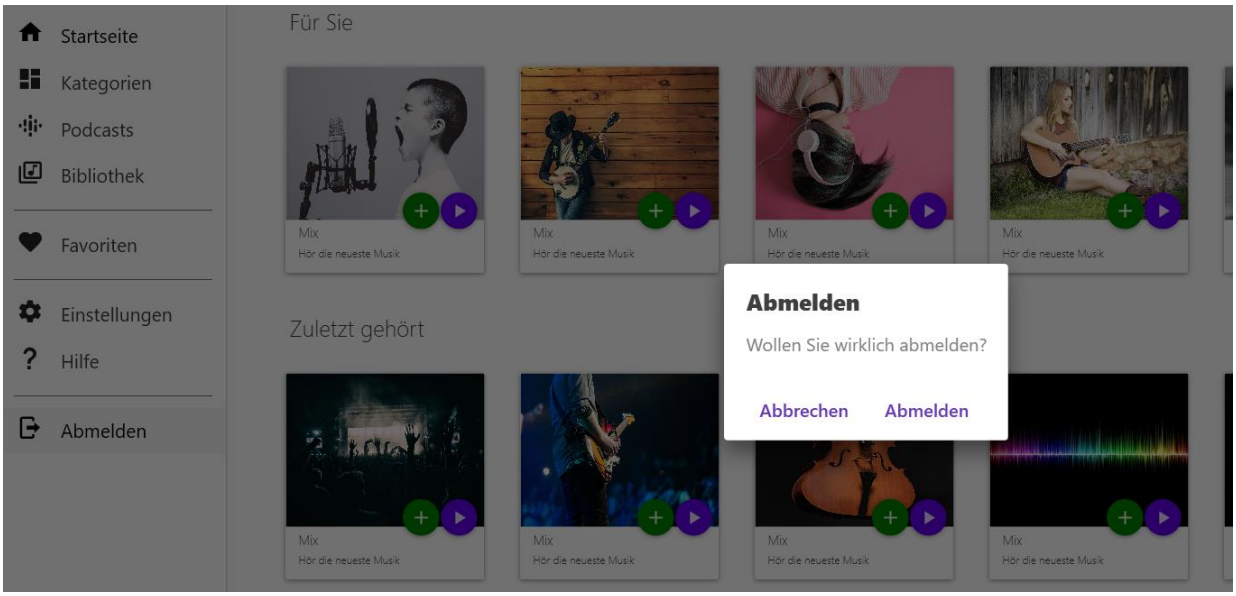
Man kann etwas in Suche-Bar eingeben und mit Enter-Taste kommt man zu Suchergebnisse.



Dropdown Menüs für Profile und Benachrichtigung.



Wenn man in der Hauptseite ist, und Abmelden geklickt hat, erscheint ein Alert-Dialog.



TECHNISCHE KONZEPTE

Das Hauptfenster enthält ein *Frame*, in dem verschiedene Seiten gezeigt werden.

```
<!--Inhalt der Seite-->
<Grid DockPanel.Dock="Bottom" >

    <Grid.RowDefinitions>
            <RowDefinition Height="*"/>
            <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <!--Inhalt Frame-->
    <Frame Panel.ZIndex="-1" BorderBrush="Transparent" x:Name="myFrame" Grid.Row="0"  NavigationUIVisibility="Hidden"
          Content="{Binding CurrentPage , Converter={local:ApplicationPageValueConverter}}" ></Frame>
```

Das Wechseln zwischen die Hauptseiten erfolgt mit Änderung der *CurrentPage* und Übergeben den neuen Zustand an dem *DataContext.*

```
    private void login(object sender, RoutedEventArgs e)
    {
       WindowViewModel.CurrentPage = ApplicationPage.MainMenu;

        ((MainWindow)Application.Current.MainWindow).DataContext = new WindowViewModel(((MainWindow)Application.Current.MainWindow));

    }
```

Das Wechseln zwischen verschachtelten Seiten erfolgt mit Änderung der *CurrentPage* und mithilfe der *NavigationService*.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    WindowViewModel.CurrentPage = ApplicationPage.Songlist;
    this.NavigationService.Navigate(new Songlist());

}
```

ApplicationPage definiert die Appseiten.

```
/// <summary>
/// Enum definiert aktuelle Seite
/// </summary>
namespace plattform
{
    public enum ApplicationPage
    {
        Login = 0,
        Register = 1,
        Startseite = 2,
        MainMenu = 3,
        Konto = 4,
        Suche = 5,
        Tab = 6,
        Search = 7,
        Kategorie = 8,
        Library = 9,
        Songlist = 10,
        Help = 11,
```

Es wird mit einem *Converter* verschiedene Seiten im *Frame* gezeigt und gesteuert.

```
using System.Globalization;

namespace plattform
{
    class ApplicationPageValueConverter : BaseValueConverter<ApplicationPageValueConverter>
    {
        public override object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            switch ((ApplicationPage)value)
            {
                case ApplicationPage.Login:
                    WindowViewModel.PlayerCard = 0;
                    return new Login();
                case ApplicationPage.Tab:
                    return new Tab();
                case ApplicationPage.Songlist:
                    return new Songlist();
                case ApplicationPage.Register:
                    WindowViewModel.PlayerCard = 0;
                    return new Register();
                case ApplicationPage.MainMenu:
                    WindowViewModel.PlayerCard = 70;
                    return new MainMenu();
                default:
                    Debugger.Break();
```

Es wird ein *WindowViewModel* verwendet, um Zugriff auf das Hauptfenster zu gewährleisten. *CurrentPage* speichert die aktuelle Seite. *SearchText* übergebt den gesuchten Text an der jeweiligen Suchergebnisseite.

```
/// Windowviewmodel für MainWindow
/// </summary>
namespace plattform
{
    public class WindowViewModel
    {
        private Window mw;

        public WindowViewModel(Window window)
        {
            mw = window;
            //Max Height, Width des Fensters werden definiert
            mw.MaxHeight = SystemParameters.MaximizedPrimaryScreenHeight;
            mw.MaxWidth = SystemParameters.MaximizedPrimaryScreenWidth;
        }
        /// <summary>
        /// CurrentPage speichet die aktuelle Seite
        /// </summary>
        public static ApplicationPage CurrentPage { get; set; } = ApplicationPage.Login;

        /// <summary>
        /// Border des Windows
        /// </summary>
        public int Border { get; set; } = 5;
        public Thickness ResizeBorderThickness{get{ return new Thickness(Border); } }

        /// <summary>
        /// Gesuchte text wird übergeben
        /// </summary>
        public static string SearchText { get; set; }

        /// <summary>
        /// Höhe der PlayerKarte
        /// </summary>
        public static int PlayerCard { get; set; } = 0;


    }
```

Es wird eine Funktion implementiert, um Abhängigkeiten der *XAML*-Strukturen in Code zu steuern (ZB. ein Kinderelement vom Elternelement).

```
using System.Windows.Media;

public static class Ext
    {
        public static T GetChildOfType<T>(this DependencyObject obj)
    where T : DependencyObject
        {
            if (obj == null) return null;

            for (int i = 0; i < VisualTreeHelper.GetChildrenCount(obj); i++)
            {
                var child = VisualTreeHelper.GetChild(obj, i);

                var result = (child as T) ?? GetChildOfType<T>(child);
                if (result != null) return result;
            }
            return null;
        }
    }
```

Material Design Pakete sind im *App.xaml* in *ResourceDictionary* verbunden.

```
<ResourceDictionary>
    <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.Light.xaml" />
        <ResourceDictionary Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.Defaults.xaml" />
        <ResourceDictionary Source="pack://application:,,,/MaterialDesignColors;component/Themes/Recommended/Primary/MaterialDesignColor.DeepPurple.xaml" />
        <ResourceDictionary Source="pack://application:,,,/MaterialDesignColors;component/Themes/Recommended/Accent/MaterialDesignColor.DeepPurple.xaml" />
```

Ein Dialog kann man mit Material design Dialoghost gestalten. Ein Dialog kann mit *CloseDialogCommand* geschlossen und mit *OpenDialogCommand* geöffnet werden.

```
<materialDesign:DialogHost.DialogContent>
    <StackPanel Margin="16">
        <Label Content="Bitte, Geben Sie Ihre E-Mail ein" ></Label>
        <StackPanel Orientation="Horizontal" Margin="0 10 0 0">
            <materialDesign:PackIcon Kind="Account" Height="20" Width="20" VerticalAlignment="Bottom" Margin="0 0 10 0"/>
            <TextBox materialDesign:HintAssist.Hint="Email" Style="{StaticResource MaterialDesignFloatingHintTextBox}"
                 VerticalAlignment="Center" Width="250" />
        </StackPanel>

        <StackPanel Orientation="Horizontal" HorizontalAlignment="Right">
            <Button Style="{StaticResource MaterialDesignFlatButton}"
                    IsCancel="True"
                    Margin="0 8 8 0"
                    Content="Abbrechen"
                    Click="Cancel"
                    Command="{x:Static materialDesign:DialogHost.CloseDialogCommand}">
                <Button.CommandParameter>
                    <system:Boolean xmlns:system="clr-namespace:System;assembly=mscorlib">
                        False
                    </system:Boolean>
                </Button.CommandParameter>
            </Button>
            <Button Style="{StaticResource MaterialDesignFlatButton}"
                    IsDefault="True"
```

Eine Snackbar kann man mit *materialDesign*-Tag gestalten. *MessageQueues* definiert eine Reihe von Snackbars mit einer Zeitperiode zwischen den Nachrichten.

```
<!-- Snackbar -->

    <materialDesign:Snackbar Margin="0 0 0 50" x:Name="Snackbar" MessageQueue="{materialDesign:MessageQueue}"   >

    </materialDesign:Snackbar>
```

PROJEKT „KOMPONENTE“

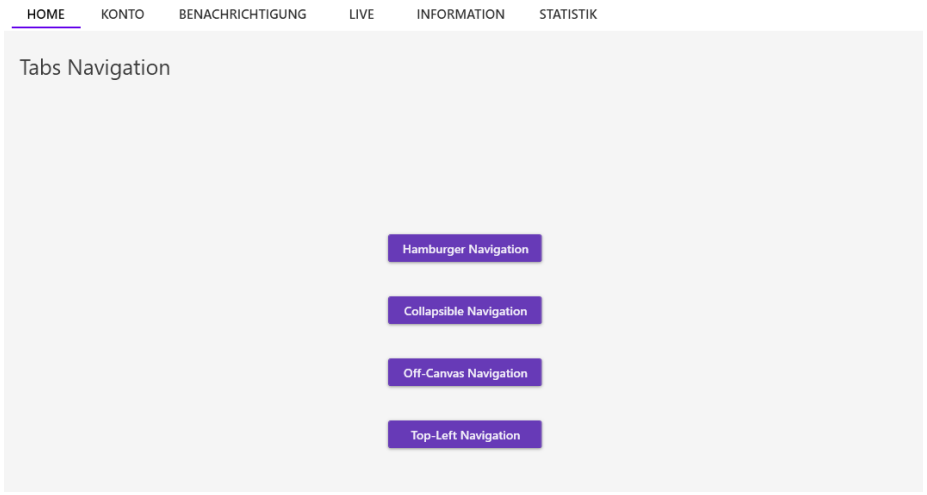
In diesem Projekt werden fünf Navigationskonzepte implementiert.

1. Tab Navigation
2. Collapsilbe Navigation
3. Hamburger Navigation
4. Off-Canvas Navigation
5. Top-and-Left Navigation

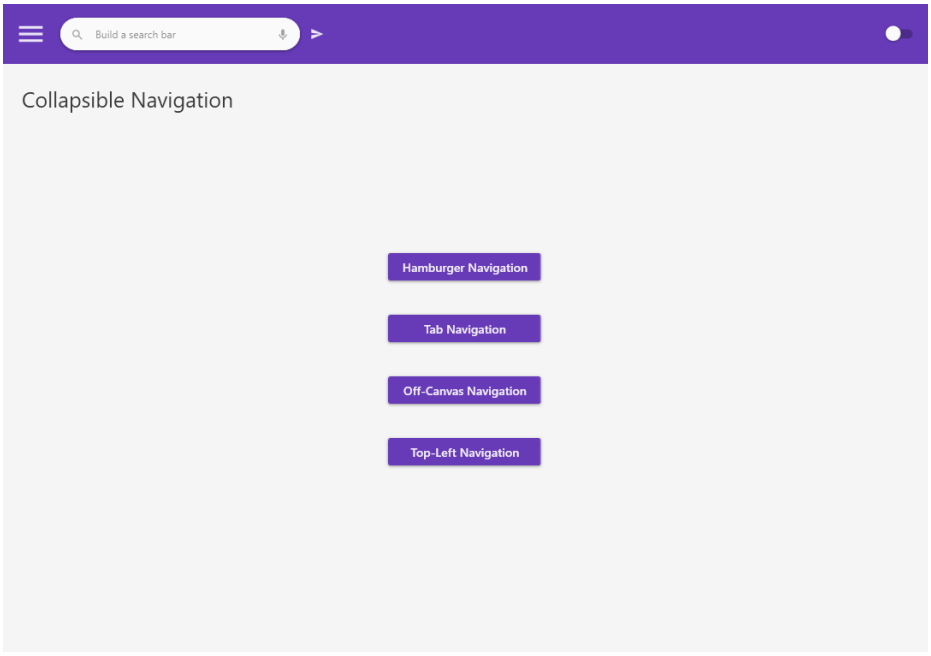
FOKUS

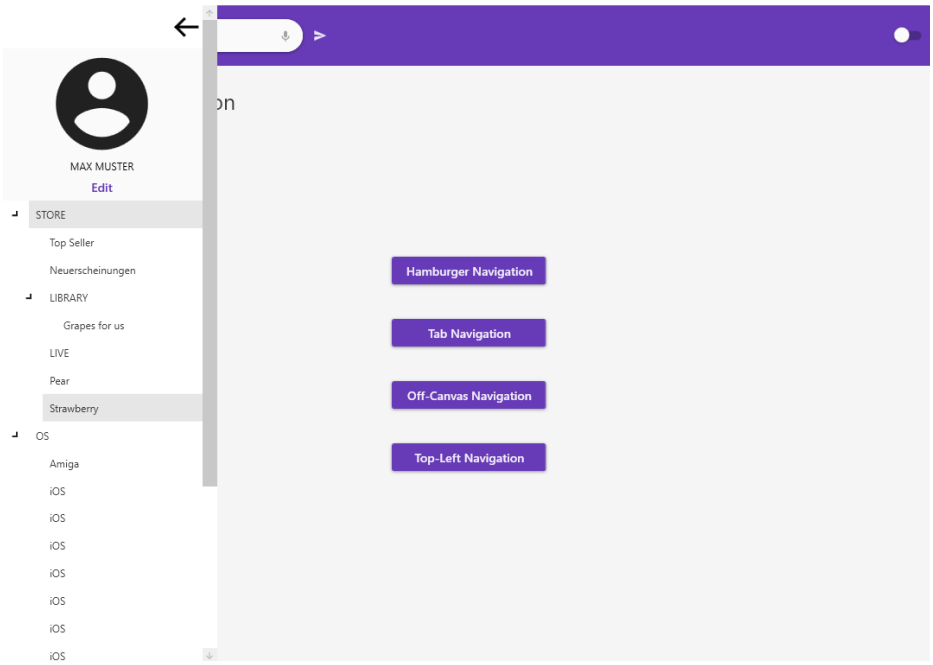
Der Fokus liegt bei der Gestaltung von verschiedenen Navigationsmöglichkeiten.

1. Tab Navigation: Tabs erlauben eine seitliche Navigation zwischen den Bildschirmen.

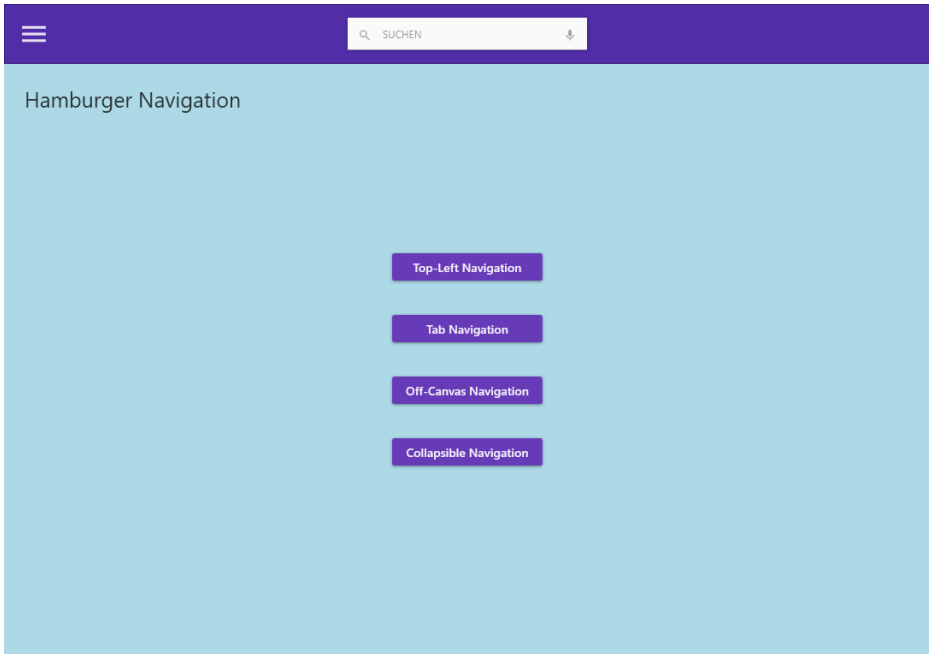


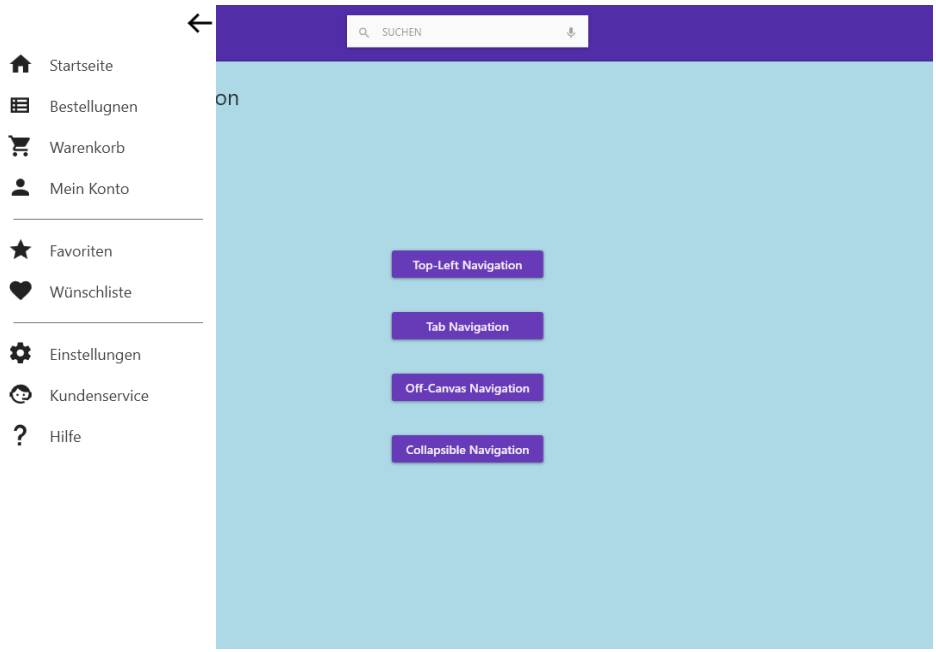
2. Collapsible Navigation: enthält ein- und aufklappbaren Teile.



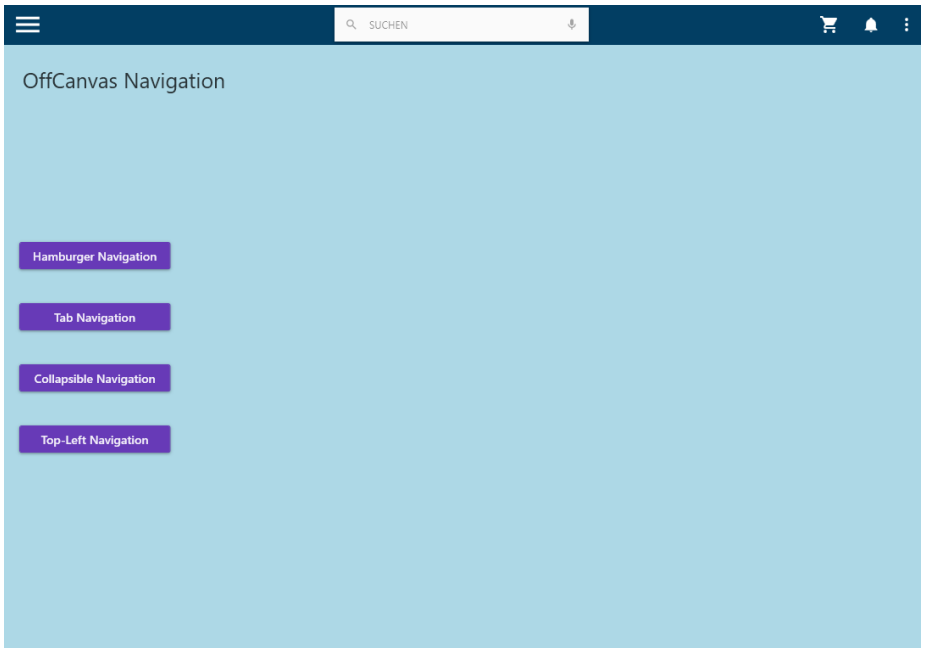


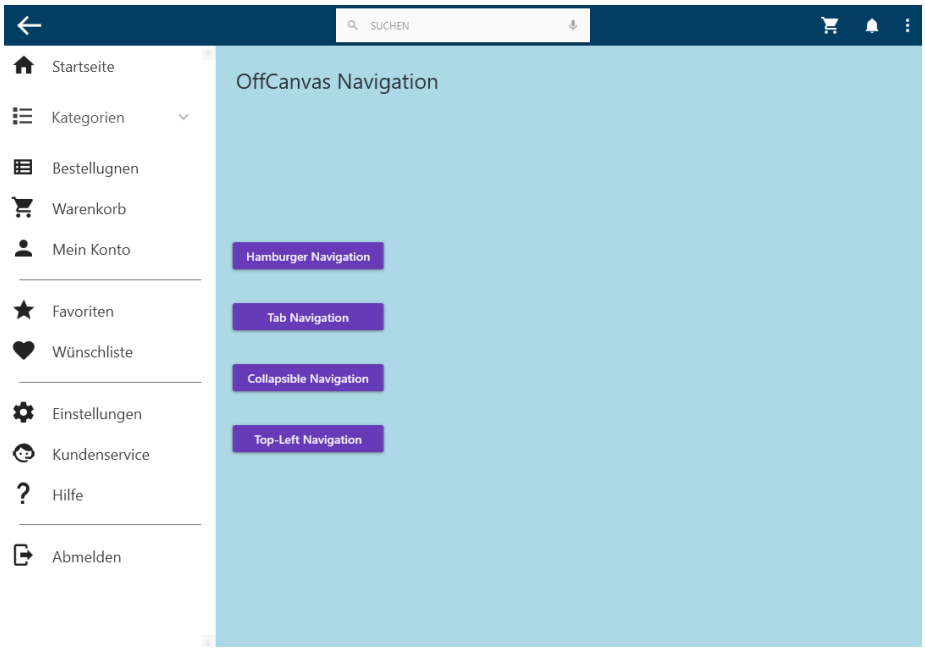
3. Hamburger Navigation: erlaubt eine seitliche Navigation und kann viele Ziele enthalten.



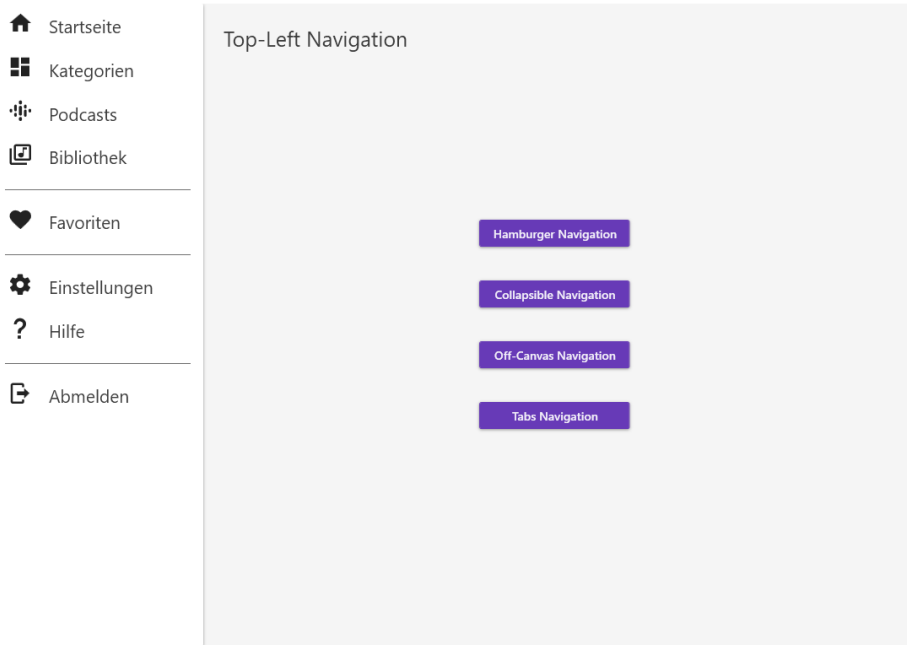


4. Off-Canvas Navigation: Hier wird beim Öffnen des Menüs die Seite nach rechts verschoben.





5. Top-and-Left Navigation: hier ist das Menü fix.



Ein *Collapsible-, Hamburger*-Menü wird in der Seite links platziert und wird am Anfang die Breite null haben. Ein Klick auf dem *ToggleButton* löst ein Event aus und wird mit *Stroyboards DoubleAnimation* die Breite von 0px auf 250px in einer eingegebenen Periode und umgekehrt bei Schließen eingestellt.

```
<!--Navigationsleiste-->
<Grid x:Name="nav_pnl"  HorizontalAlignment="Left" Width="0"
      Background ="White"
      ZIndex ="2">

    <ScrollViewer PreviewMouseWheel="ScrollViewer_PreviewMouseWheel" Panel.ZIndex="1"
                  VerticalScrollBarVisibility="Visible" HorizontalScrollBarVisibility="Disabled" >
        <Grid>
            <StackPanel >

                <DockPanel Height="50" VerticalAlignment="Top" HorizontalAlignment="Right">
                    <ToggleButton x:Name="tgl2"   Style="{DynamicResource MaterialDesignHamburgerToggleButton}" HorizontalAlignment="Right"
                                  Checked="tgl2_Checked" Unchecked="tgl2_Unchecked" >
                        <ToggleButton.Triggers>
                            <EventTrigger RoutedEvent="ToggleButton.Unchecked">
                                <BeginStoryboard>
                                    <Storyboard x:Name="storyBoardClose">
                                        <DoubleAnimation Storyboard.TargetName="nav_pnl"
                                                BeginTime="0"
                                                Duration="0:0:0.3"
                                                Storyboard.TargetProperty="Width"
                                                From="250" To="0" >

                                        </DoubleAnimation>
                                    </Storyboard>
                                </BeginStoryboard>
                            </EventTrigger>
                            <EventTrigger RoutedEvent="ToggleButton.Checked">
                                <BeginStoryboard>
                                    <Storyboard>
                                        <DoubleAnimation Storyboard.TargetName="nav_pnl"
                                                BeginTime="0"
                                                Duration="0:0:0.3"
                                                Storyboard.TargetProperty="Width"
                                                From="0" To="250" >

                                        </DoubleAnimation>
                                    </Storyboard>
                                </BeginStoryboard>
```

Ähnliches gilt für *Off-Canvas* Menü. Allerdings ist das Menü hier Teil des Seiteninhalts und so wird beim Öffnen und Schließen den Inhalt der Seite nach recht bzw. Nach links verschoben.

```
<!--Rest der Seite-->
<Grid Grid.Row="1">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
    <!--Navigationsleiste-->
    <Grid x:Name="nav_pnl"  HorizontalAlignment="Left" Grid.Row="1" Width="0"
      Background ="White"
      ZIndex ="2">

        <ScrollViewer HorizontalScrollBarVisibility="Hidden" PreviewMouseWheel="ScrollViewer_PreviewMouseWheel" >
            <StackPanel Orientation="Vertical" >


                <ListView FontSize="18">
                    <ListViewItem   >
                        <StackPanel Orientation="Horizontal">
                            <materialDesign:PackIcon Kind="Home" Height="30" Width="30" />
                            <Label Content="Startseite" Margin="15 0 0 0" VerticalAlignment="Bottom"></Label>
                        </StackPanel>

                    </ListViewItem>
                    <ListViewItem >
                        <Expander Background="Transparent"  Width="230" Margin="10 0 0 0">
                            <Expander.Header>
                                <StackPanel Orientation="Horizontal" HorizontalAlignment="Left">
                                    <materialDesign:PackIcon Kind="FormatListBulletedSquare" Margin="-35 0 0 0" Height="30" Width="30"/>
                                    <Label Content="Kategorien" Margin="10 0 0 0"></Label>
                                </StackPanel>
```

Die Tabs werden mit *RadioButtons* implementiert und der Inhalt des Tabs kann in einem Frame gezeigt werden. Man kann durch die Tabs horizontal scrollen, wenn die Breite des Fensters klein wird.

```
<!--Tabs-->
<Border Grid.Row="0" BorderThickness="0.5" >
    <Grid  Height="40" Background="#FFFFFF" >


        <ScrollViewer x:Name="scrollerTab" VerticalAlignment="Top" HorizontalScrollBarVisibility="Hidden"
                      VerticalScrollBarVisibility="Disabled" Foreground="Black" PreviewMouseWheel="ScrollViewer_PreviewMouseWheel">
            <WrapPanel Orientation="Horizontal" Margin="10 0 0 0">

                <RadioButton x:Name="firstTab" Style="{StaticResource MaterialDesignTabRadioButton}"
                    Margin="4"
                    Checked="RadioButton_Checked"
                    IsChecked="True"
                    FontSize="15"
                    Content="HOME" />
                <RadioButton Style="{StaticResource MaterialDesignTabRadioButton}"
                    Margin="4"
                    Checked="RadioButton_Checked"
                    IsChecked="False"
                    FontSize="15"
                    Content="KONTO"  />

                <RadioButton Style="{StaticResource MaterialDesignTabRadioButton}"
                    Margin="4"
                    Checked="RadioButton_Checked"
                    FontSize="15"
                    IsChecked="False"
                    Content="BENACHRICHTIGUNG" />
                <RadioButton Style="{StaticResource MaterialDesignTabRadioButton}"
                    Margin="4"
                     Checked="RadioButton_Checked"
                    FontSize="15"
                    IsChecked="False"
```