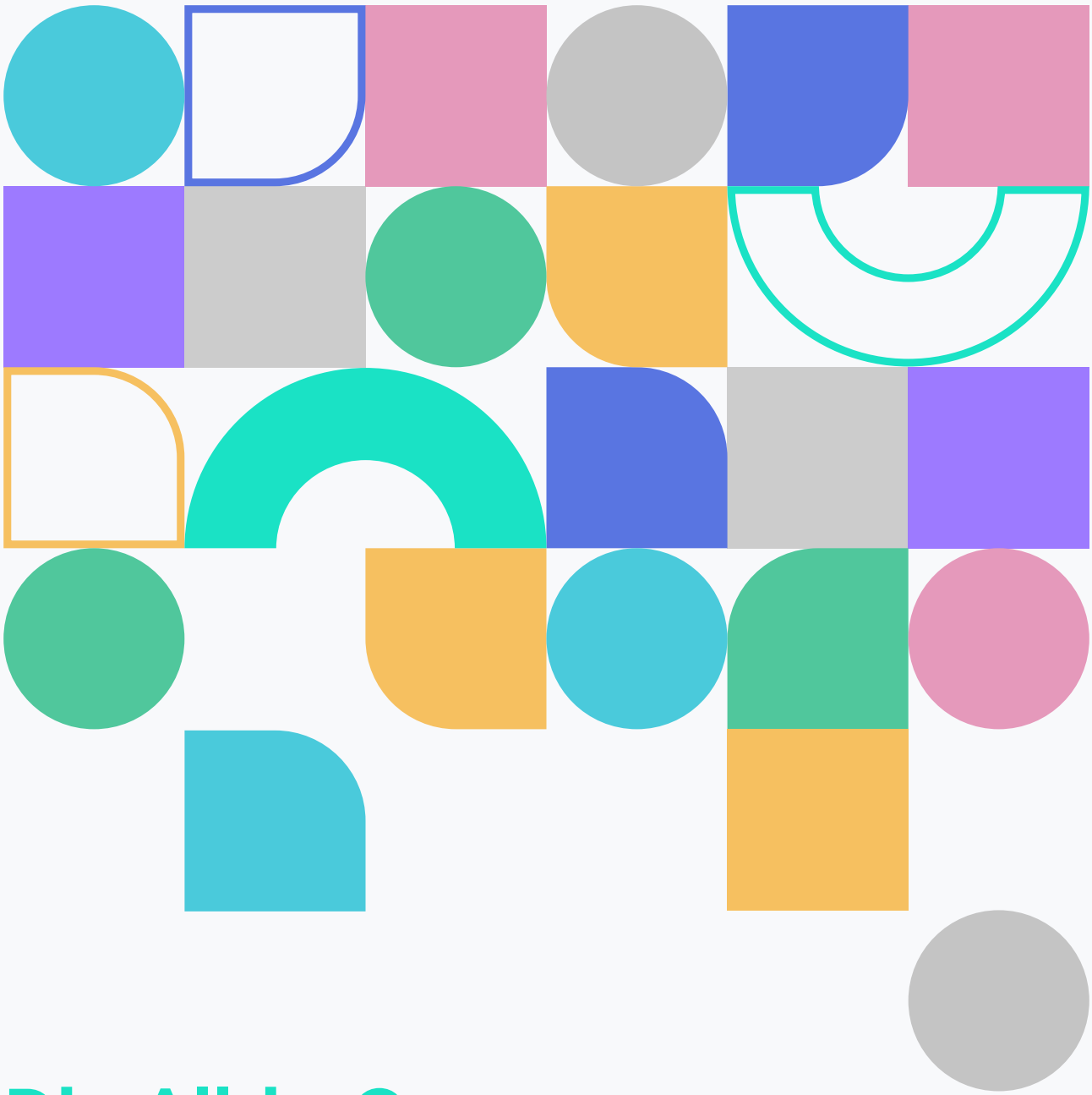


Die All-in-One-Web-App für die Verbesserung deines Wohlbefindens

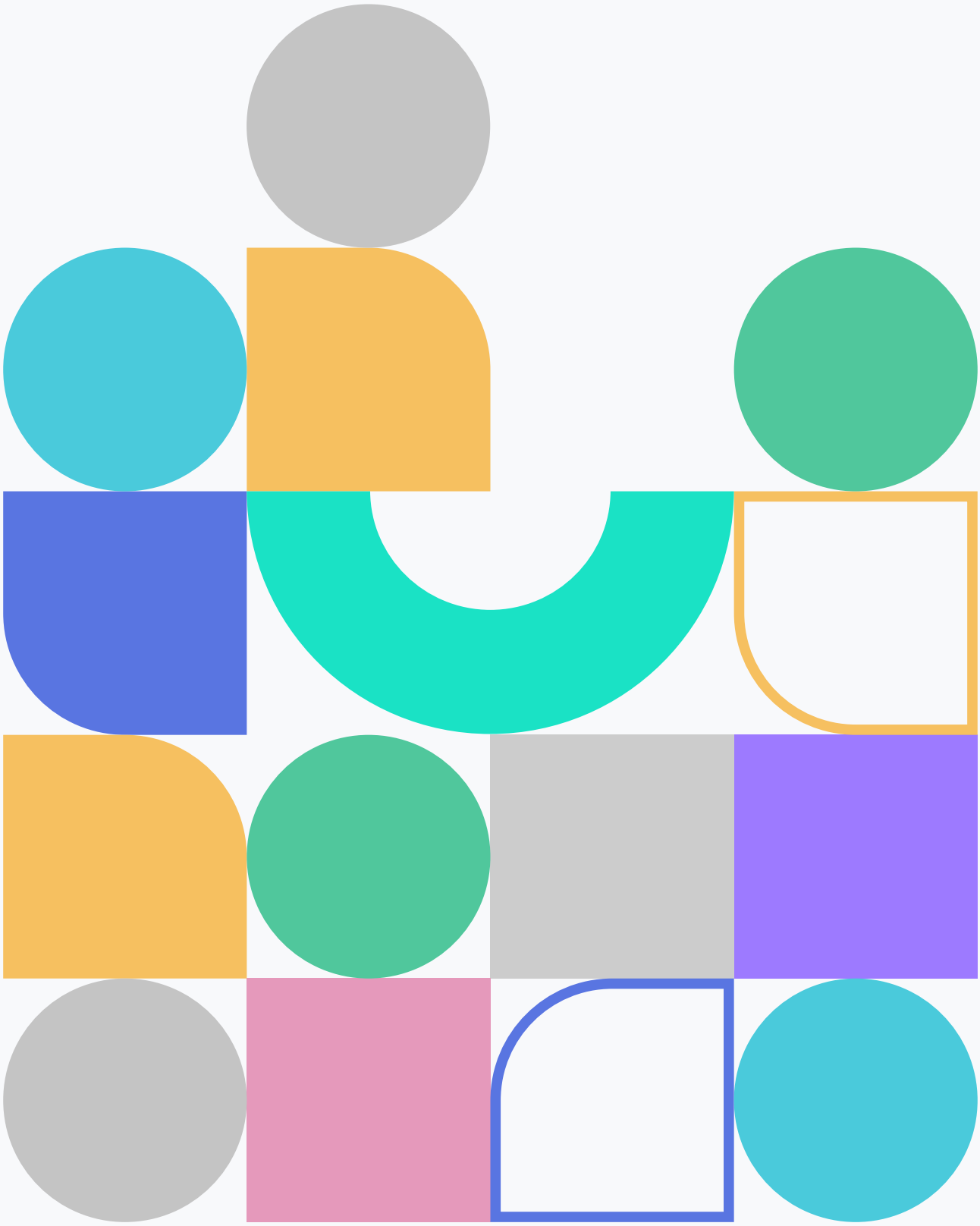
MASTERPROJEKT - PRODUKTIONSSEMESTER

Interaktive Mediensysteme, Websystems
Benjamin Adolph | Jennifer Konopka | Veronika Remiger | Sarah Schneller

Hochschule Augsburg
University of Applied Sciences

INHALTSVERZEICHNIS

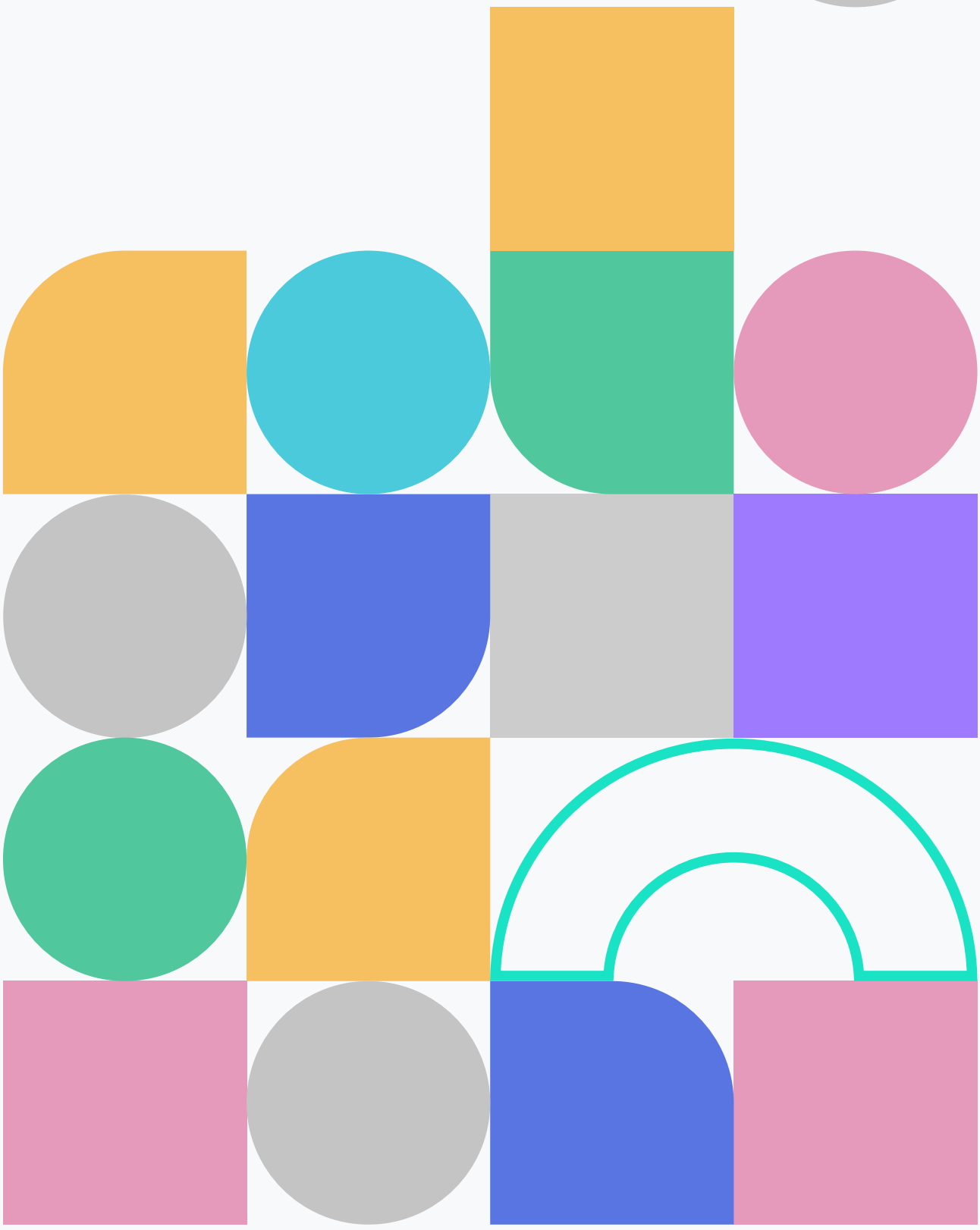
1. ABSTRACT



Immer mehr junge Menschen leiden unter chronischen Krankheiten. Sie müssen lernen, mit diesen umzugehen und das ihr Leben lang. Immer mehr Menschen erkranken an Krankheiten, die einen ein Leben begleiten. Umso wichtiger ist es, sich selbst und seine Krankheit kennenzulernen, um sich auf jede Situation einzustellen und zu wissen, wie der Körper funktioniert. Die App fine dient chronisch Kranken ihre Beschwerden im Überblick zu behalten und anhand eines digitalen Tagebuchs zu dokumentieren, welche Zusammenhänge zwischen Maßnahmen und Symptomen bestehen. So können Schmerzen schneller gelindert und schlimme Reaktionen auf alltägliche Dinge womöglich verhindert werden. Die Dokumentation der Symptomatik ist Kernbestandteil von fine und soll Lösungsmöglichkeiten aufdecken. Eine medizinische Beratung ist jedoch nicht gegeben. Hierfür können die Daten und Aufzeichnungen ganz einfach beim nächsten Arztbesuch gezeigt werden.

Fine ist darauf ausgelegt, viele verschiedene Krankheitsbilder zu bedienen, was durch den individuellen und modularen Aufbau bezweckt wird. Im Gegensatz zu den schon bestehenden analogen Tagebüchern ermöglicht fine ein unkompliziertes und schnelles Eintragen sowie die Möglichkeit auch in die Tiefe bis ins kleinste Detail zu dokumentieren. Je nach Bedürfnis kann fine flexibel angepasst werden. Zudem soll nicht nur das Smartphone bedient werden, sondern auch das Tablet. Je nach Endgerät ändert sich die Anzeige, aber die Funktionen bleiben gleich.

2. RÜCKBLICK KONZEPTIONS-SEMESTER



Das übergeordnete Ziel, chronisch kranke Menschen zu unterstützen, sich und ihre Krankheit besser zu verstehen, gilt nach wie vor als oberste Priorität. Im letzten Semester stand in erster Linie das Konzipieren dieses Projektes zur Vorbereitung auf die Umsetzung. Die Zielgruppe wurde eingegrenzt, Marktforschung betrieben, Marke und System definiert, der erste Styleguide stand und auch in der Technologie wurde geschaut, wie man am besten vorgehen kann. Diese Vorbereitung vor allem der technischen Seite hat insofern sehr geholfen, dass wir im zweiten Semester bei der Umsetzung nicht lange überlegen mussten und direkt anfangen konnten. Bereits in den Semesterferien stand die komplette Entwicklungsumgebung.

Die schier hohe Anzahl an Modulen ist seitdem noch einmal auf die wesentlichen sechs zurück geschrumpft. Dies erleichtert besonders im Design viele Dinge und einige der zuvor konzipierten Module wie Arztbesuche und Umwelteinflüsse können anders in die App bei Bedarf integriert werden.

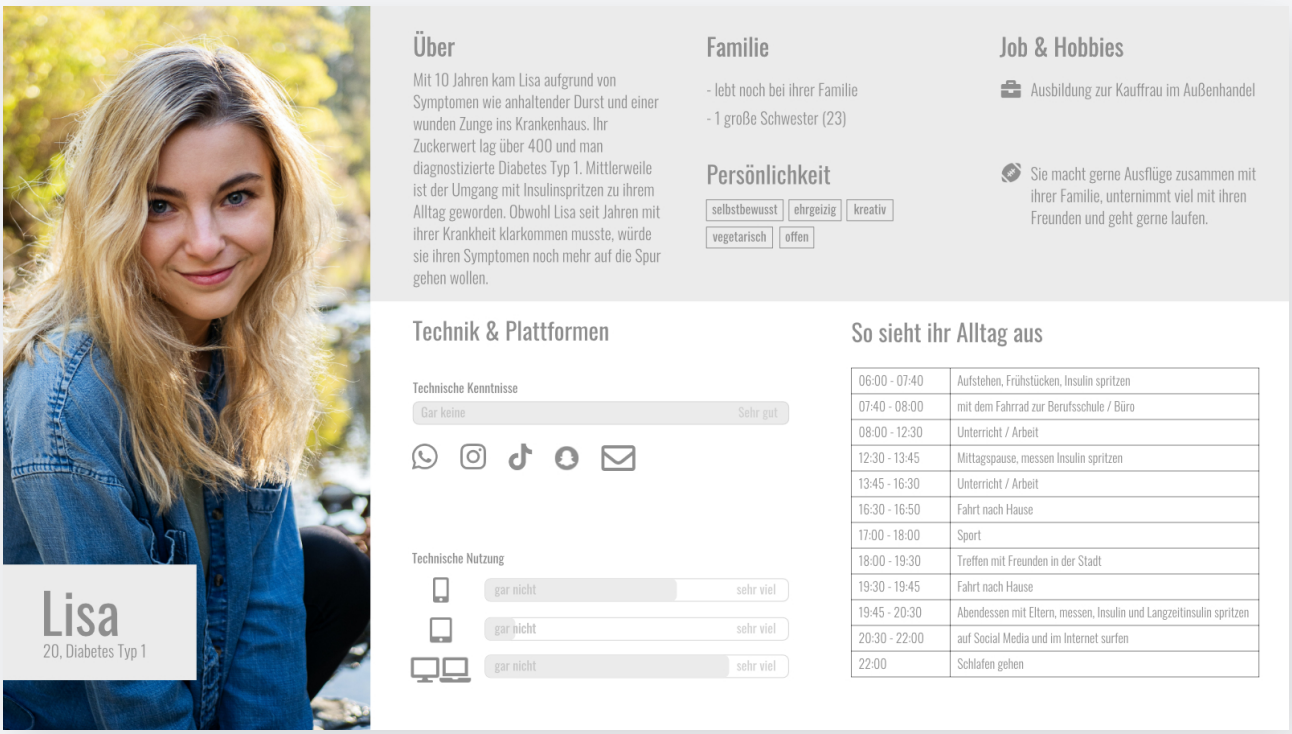


Abb. 1: Personal Lisa

Insgesamt sechs Personas haben wir letztes Semester kreiert. Noch bis heute sind sie ausschlaggebend für die Bestimmung unserer Zielgruppe. Gerade im Konzept und Design muss stets auf die Bedürfnisse der Zielgruppe geachtet werden. Wo wir uns im letzten Semester nur um die mobile Variante Gedanken gemacht haben, kümmern wir uns im zweiten Semester auch um die Forderung nach einer Tablet-Variante. Unsere Zielgruppe ist offen für Neues, hat keine Angst vor Veränderungen und geht auch mal ein Risiko ein. Dies sind wichtige Grundbausteine für die weitere Entwicklung der App fine und enorm wichtig, wenn es um die Formensprache, die verwendeten Elemente und Beschriftungen sowie die Screendesigns geht.

3. PROJEKTMANAGEMENT

Auch in diesem Semester galt es, Tickets zu sortieren, einen ersten Zeitplan aufzustellen und sich eine Übersicht über alle Aufgaben zu verschaffen. Jeder war für seinen eigenen Bereich zuständig und erstellte dort Tickets. Dennoch war von Beginn des Semesters klar, dass das Produktionssemester anders gehandhabt werden muss. Im Folgenden sind Zeitplan sowie Meilensteine, angewandte Methoden und Meetings aufgeführt.

3.1. ZEITPLAN UND MEILENSTEINE

Auf Miro wurde fortgeführt, was im letzten Semester bereits begonnen wurde. Mithilfe eines Boardes sind alle Meilensteine für die Umsetzung aufgelistet und zeitlich eingeordnet worden. Dabei handelt es sich um Bereiche im Design, vor allem Screendesigns auch zum Tablet, um die Entwicklungsumgebung, das Backend, Frontend und neue Konzepte wie etwa zum Sprachassistenten oder der Handhabung von Offline-Funktionalität. Auch Interviews und Nutzertests wurden berücksichtigt.

Im Laufe des Semesters wurde klar, dass nicht alle Ziele gleichermaßen erreicht werden können, sodass besondere Features als “Nice to have” eingeordnet wurden. Prioritäten wurden verteilt und um hier ein besseren Überblick zu behalten, wurde entschieden zunächst Trello als Tool hinzuzufügen. Jedoch zeigte sich das Ausmaß an Tickets bereits nach sehr kurzer Nutzung, sodass schließlich ein Wechsel zu Jira nötig war. In Jira gibt es die Möglichkeit, nur einen Teil der Tickets im aktiven Sprint anzuzeigen und den Rest in den Backlog zu schieben. Ab hier fand zudem ein Wandel in der Handhabung der Meetings und des gesamten Projektmanagements statt. Wir haben gemerkt, wie wichtig es ist, eine Person als Projektmanager zu beauftragen, die über alles einen Überblick hat und für Motivation im Team sorgt. Deshalb führte dies zur Entscheidung einen agilen Ansatz zu nutzen.

3.2. METHODEN

Das Projektmanagement Tool Jira von Atlassian ermöglicht es uns, wöchentliche Sprints durchzuführen. Ein Vorteil von Jira gegenüber Trello ist vor allem die Möglichkeit, Sprints übersichtlich zu planen und Epics anzulegen, die als Meilensteine visualisiert werden können. An die Scrum-Methode angelehnt traf sich das Team, um zu besprechen, wie der letzte Sprint verlief, wie der aktuelle Stand bei Aufgaben ist und was im nächsten Sprint alles ansteht. Um relativ flexibel zu bleiben, konnten neue Aufgaben in den aktuellen Sprint dazugeholt werden, sofern ein Teammitglied über freie Kapazitäten verfügt. Ebenso konnten Aufgaben in den nächsten Sprint mitgenommen werden, wenn es nicht möglich war, die Aufgabe kleinteiliger einzuteilen bzw. diese über die Woche zu erledigen. So wurde iterativ und agil im Team am Projekt gearbeitet. Die Rollenverteilung lautet wie folgt:

Benjamin Adolph
Backend / Frontend

Jennifer Konopka
Frontend / Design

Sarah Schneller
Projektmanagement / Frontend

Veronika Remiger
Design / Frontend

3.3. MEETINGS

Am Donnerstag fanden Sprint-Review, Sprint-Retrospective und Sprint-Planning in einem Meeting statt. So wurde besprochen, was alles im letzten Sprint geschafft wurde und welche Probleme auftauchten, was als nächstes ansteht und wo wir Verbesserungen sehen sowie die neue Verteilung von Tickets und wenn noch nötig eine Absprache, ob ein bestimmter Task als “fertig” einzuordnen ist. Da die Sprints einwöchig aufgeteilt waren, kam es auch nicht zu allzu langen Besprechungen. Neben dem Meeting am Donnerstag gab es jeden Dienstag ein Treffen über Zoom später auch vor Ort in der Hochschule.

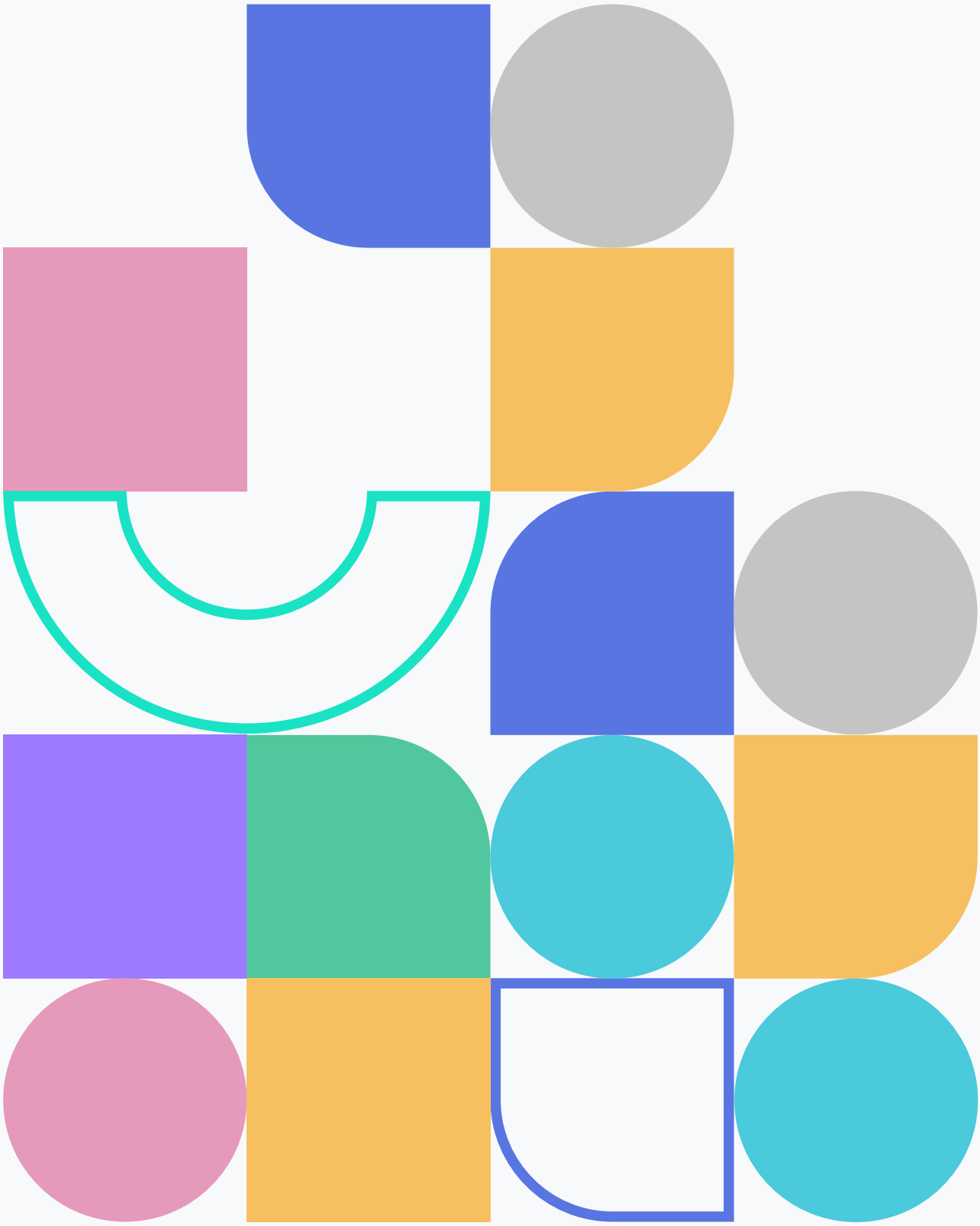
Je nachdem, ob gemeinsam an großen Aufgaben gearbeitet oder wichtige Punkte besprochen werden sollten, war dieser Termin frei gestaltbar. Generell war hier Zeit für den Product Backlog und sonstige Absprachen. Zusätzlich gab es sonntags ein Weekly, an dem der Stand der Dinge kurz besprochen wurde. Sowieso war das Team sehr flexibel. Wenn es ein Problem gab und man sich spontan treffen wollte, fand man zusammen schnell zu einer Lösung. Auch Absprachen zwischen einzelnen Personen oder Zusammenarbeit zu zweit an Aufgaben wurde leicht bewältigt.

3.4. ZIELSETZUNG

Als Ziel ist ein lauffähiger Prototyp festgelegt worden, der über eine Eingabe der Symptome, einen Kalender, einen User Login sowie eine Schnelleingabe im Dashboard verfügen sollte. Außerdem sollte man sich die Module zu Beginn der Nutzung der Anwendung zusammenstellen können. Zudem sollten Daten mit dem Backend verknüpft werden.

Das führte dazu, dass wir besonders am Anfang des Semesters mit Problemen zu kämpfen hatten. Es wurden lange Diskussionen geführt, Absprachen gemacht und es war häufig nur das große Ganze im Kopf, was es erschwerte, sich in spezifische Bereiche hineinzuversetzen und diese herunterzubrechen und mit dem Umsetzen zu beginnen. Dies waren keine Probleme technischer Art, sondern es ging häufig um die Struktur und Vorgehensweise.

4. KONZEPT UND DESIGN



Aufbauend auf den Stand des letzten Semesters sind Konzept und Design weiter angepasst und ausgebaut worden. In diesem Abschnitt gehen wir zunächst auf die zu Beginn des Semesters geführten Experteninterviews ein, an denen wir uns bei einigen Design- und Konzeptfragen orientiert und darüber diskutiert haben. Danach folgen Designentscheidungen und Vorgehensweisen zum Styleguide, dem Branding, dem Designsystem, UX Writing, Screendesign und schließlich dem Marketing.

4.1. EXPERTENINTERVIEWS

Um uns zu vergewissern und zu überprüfen, ob unser Konzept aufgeht, haben wir Experten im Bereich der chronischen Krankheiten befragt und Interviews geführt. Anhand erster Screendesigns wurde ein Leitfaden erstellt, an dem wir uns für die einzelnen Module orientieren konnten. Insgesamt fünf Personen, darunter eine Assistenzärztin, ein Orthopäde und drei Medizinstudenten, haben sich zu den Interviews bereit erklärt.

Die Ergebnisse zeigen, dass das Modul Symptome ganz richtig den höchsten Stellenwert in der Umsetzung erhält. Außerdem wurde unser Konzept bestätigt, da man uns sagte, dass diese Anwendung den Leuten das Gefühl geben könnte, dass sie die Kontrolle über ihr Leben haben. Dies muss gegen Ende Semesters in der Testphase genauer überprüft werden. Ganz wichtig, war auch die Aussage, dass Hausärzte dieses Vorhaben unterstützen würden, da Patienten sich so selbst besser kennenlernen könnten. Es helfe bei einer Therapie noch eher als bei einer Diagnose, würde aber dazu führen, dass Patienten dann auch bei der Therapie gewissenhafter mitmachen und mehr folgsam sind, die Therapie durchzuführen. Schließlich ist noch zu erwähnen, dass die Anwendung auch auf Tablets und am Desktop bedienbar sein sollte.

Die von uns konzipierten Module fanden fast alle Resonanz. Es wurde sofort klar, dass die Umwelteinflüsse nur in geringem Maße wirklich medizinisch von Bedeutung waren. Da man bei diesem Modul auch nichts selbst eintragen kann, sondern nur das Wetter und die Mondphasen angezeigt bekommt, haben wir uns entschlossen, diese im Kalender direkt an den Tagen anzuzeigen. Außerdem haben wir die Mondphasen gänzlich entfernt.

Das Feedback für alle weiteren Module war sehr breit gefächert und hilfreich. An dieser Stelle gehen wir spezifisch auf die umgesetzten Module Symptome und Gefühle ein. Bei den Symptomen war sofort klar, dass die Häufigkeit und Intensität eine sehr große Rolle spielen. Normalerweise ist es gängig, seinen Schmerz auf einer Skala von eins bis zehn einzuordnen. Hier haben wir uns entschieden eine einheitliche Skala über alle Module hinweg zu verwenden, sodass Einträge immer von eins bis fünf getätigt werden können. Des Weiteren wurden wir darauf hingewiesen, dass alle W-Fragen sehr wichtig sind: Wo?, Seit wann?, Was?, Wieso?. So kamen wir schließlich zum einstimmigen Konsens, dass gerade bei der Lokalisierung der Symptome eine Figur vorliegen muss, an die herangezoomt werden kann. So kann eine bestimmte Stelle an Hand, Bein oder Kopf besser ausgewählt werden.

Bei den Gefühlen steht an erster Stelle, später auch einen Weg zu finden, wie soziale Aspekte besser integriert werden können. Das Bio-Psycho-Soziale Modell beinhaltet sowohl die Psyche als auch das soziale Umfeld. Dies meint, wie auch Abbildung 1 zu entnehmen, Arbeits- und Wohnsituation als auch Familie. Hier geht es darum, ob jemand Beziehungs-

probleme hat und wie viel Unterstützung derjenige erhält. Fragen wie “Nimmt der Partner die Krankheit ernst?” werden hier geklärt und sind, wie im Modell erkennbar, essentiell für ein gutes Gefühl bzw. eine gute Stimmung.[0]

Es gab noch viele weitere gute Tipps. Würde es zu einer Iteration mit Nutzertests und anschließender Umsetzung der Verbesserungen kommen, könnte erneut auf die Experteninterviews zurückgegriffen werden. Denn besonders bei den Modulen, die noch nicht umgesetzt worden sind, gibt es noch jede Menge Diskussionsbedarf auch anhand der Designs.

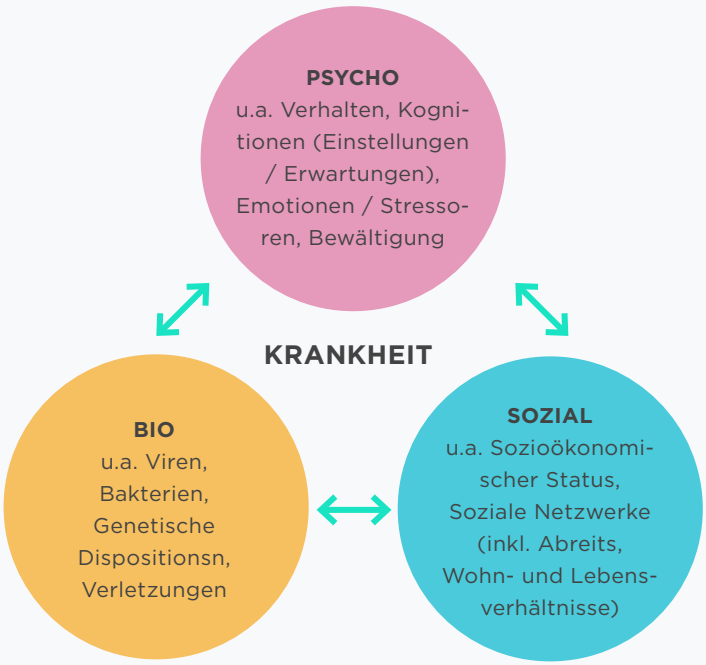


Abb. 2: Bio-Psycho-Soziales Modell modifiziert nach Engel 1977

4.2. STYLEGUIDE

4.2.1 Farben

Die Farbe Blau wirkt beruhigend, steht für Innovation und Technik und strahlt Zuverlässigkeit, Loyalität, Ruhe und Vertrauen aus. Grün ist die Farbe der Hoffnung und steht für Gleichgewicht, Leben, Harmonie und Gesundheit. Die Farbe wirkt erholsam, heilend, ruhig und entspannend. Da die Eigenschaften beider Farben sehr gut mit den Bedürfnissen unserer Zielgruppe übereinstimmen, haben wir uns für ein sehr intensives Türkis als Brandfarbe entschieden. Damit sich unsere Marke von anderen abhebt und heraussticht, ist die Farbe sehr intensiv und wird hauptsächlich als Akzent verwendet.

Wir verwenden für Typografie und Gestaltungselemente verschiedene Grautöne, da diese neutral und zurückhaltend wirken. Zudem verzichten wir auf die Farbe Schwarz, da sie mit Tod, Trauer und Depressionen in Verbindung gebracht werden kann.

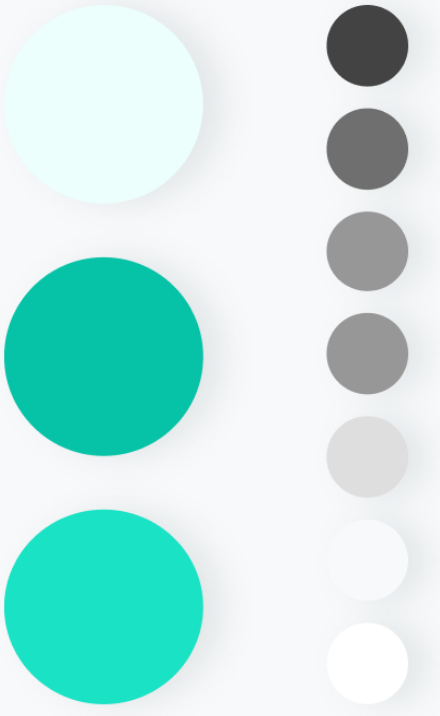


Abb. 3: Hauptfarben

Um Module schnell voneinander unterscheiden zu können, hat jedes Modul eine eigene Farbe, die von der Bedeutung gut zum Inhalt passt. Beispielsweise haben wir dem Modul Schlaf Violett zugeordnet, da diese Farbe für Phantasie und Mystik stehen kann. Die große Schwierigkeit bestand darin, sechs unterschiedliche Farben zu finden, die gut zusammenpassen, aber nicht kindlich wirken.

Um zu gewährleisten, dass jeder die Modul-Farben gut voneinander unterscheiden kann, haben wir sie auf verschiedene Farbenblindheiten getestet. Dadurch haben wir einzelne Nuancen nochmals abgeändert und optimiert.

Abb. 4: Modulfarben

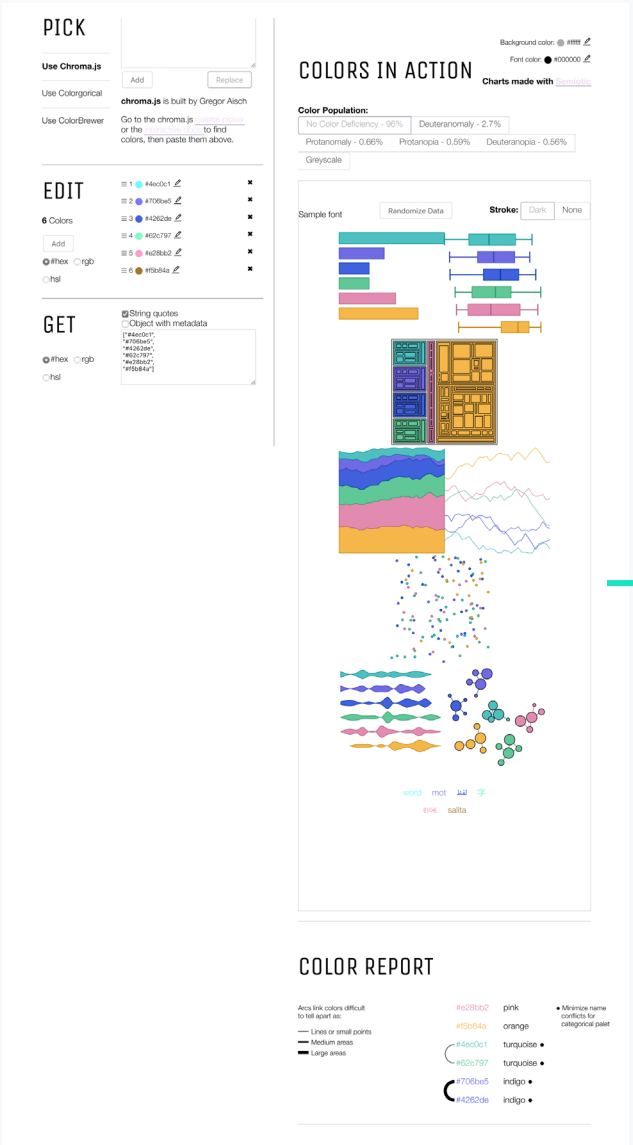


Abb. 5: Farbtest vorher

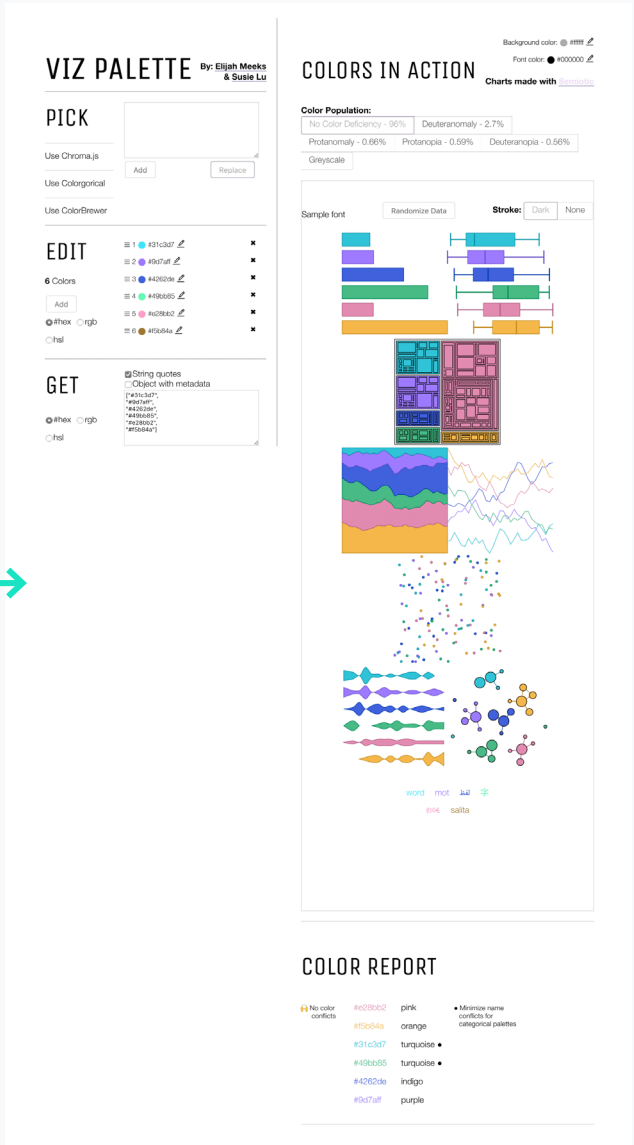


Abb. 6: Farbtest nachher

4.2.2 Formensprache

Passend zum Logo verwenden wir hauptsächlich eine geometrische und reduzierte Formensprache. Dadurch entsteht ein ruhiger, angenehmer Eindruck. Inhalte werden durch klare Formen voneinander getrennt, damit man sie besser voneinander abgrenzen kann. Durch die Verwendung von Verläufen und Schattierungen erzeugen wir einen dreidimensionalen Effekt. Zur Abtrennung von Inhalten verwenden wir dünne, unauffällige Linien und Farbflächen.

4.2.3. Typographie

Scale Category	Typeface	Weight	Size	Case	Letter spacing	Use for
Plain/L	Gotham	Medium	22	Sentence	0	Category
Plain/XM/Bold	Gotham	Bold	18	Sentence	1,5	Voice input headline
Plain/XM/Book	Gotham	Book	18	Sentence	0,5	Voice input text
Plain/M/Bold	Gotham	Bold	14	Sentence	1,5	Buttons, Headline
Plain/M/Book	Gotham	Book	14	Sentence	0,5	Subheadline
PLAIN/S/BOLD	Gotham	Bold	11	Upper case	2	Caption
Plain/S/Book	Gotham	Book	11	Upper case	2	Time

Die Schriftart Gotham haben wir gewählt, da sie für unsere Zielgruppe sehr gut geeignet ist. Sie ist modern, aber nicht verspielt. Durch meist gleichmäßige Rundungen und Strichstärken hat sie ein einheitliches und ruhiges Erscheinungsbild. Dadurch können sich die Nutzer auf den Inhalt konzentrieren und werden nicht abgelenkt. Durch den modernen Charakter der Gotham und ihre reduzierte Formensprache passt sie sowohl zu unserer jungen Zielgruppe (Elisa / David), als auch zu unserer älteren (Jonas).

Unsere User werden die Anwendung hauptsächlich auf ihrem Smartphone in der App bedienen. Dabei verwenden wir kaum Fließtexte, sondern viele kurze Über- oder Unterschriften. Deshalb ist das größte Entscheidungskriterium für die Auswahl der Schriftart die Lesbarkeit bei kurzen Texten und kleinen Schriftgrößen.

Die Gotham hat eine hohe X-Höhe und Laufweite. Dadurch sind die Buchstaben weiter voneinander entfernt und die Minuskeln

verhältnismäßig groß. Dies führt dazu, dass Texte mit kleinen Schriftgrößen besser entziffert werden können. Eine zu hohe Laufweite verschlechtert bei langen Texten die Lesbarkeit. Da wir bei fine allerdings hauptsächlich kurze Texte verwenden, ist eine hohe Laufweite besser geeignet. Für lange Texte kann die Laufweite angepasst werden.

Ein weiterer Vorteil der Gotham ist die Möglichkeit zur Konfiguration von bestimmten Ziffern (a, 3, Q). Um ein einheitliches Erscheinungsbild zu erzeugen und eine gute Lesbarkeit zu gewährleisten, haben wir uns für runde Formen entschieden.

ABCČĆDĐEFGHIJKLMNOPQRSŠTUV
WXYZŽabcčćdđefghijklmnopqrsštu
vwxyzžАБВГҐДЂЕЁЄЖЗЅИІЇЙЈКЛЉ
МНЊОПРСТЋУЎФХЦЧЏШЩЪЫЬЭ
ЮЯабвгґдђеёєжзѕиіїйјклљмнњопр
стћуўфхцчџшщъыьэюяАВГДЕZНΘІ
ΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩαβγδεζηθικλμ
νξοπρστυφχψωάΆέΈέΉίΐΊΐόΌύΰϋΥΫ
ΏĂÂÊÔăâêô1234567890‘?’“!”(%)[#]
{@}/&\<-+÷×=>®©$€£¥¢:;,.*

4.2.4. Branding

Abb. 7: Logo fine

Abb. 8: Gestaltungselemente

Logo

Das Logo besteht aus der Wortmarke „fine“ und einer Bildmark in Form eines abstrakten Lächelns. Das soll die Verbesserung des Wohlbefindens, also das Ziel von fine, darstellen. Es ist in unserer Hauptfarbe eingefärbt. Da unsere System-Persona organisiert, sachlich und vertrauenswürdig ist, haben wir das Logo sehr minimalistisch und professionell gestaltet. Dadurch hat es auch eine sehr gute Fernwirkung.

Gestaltungselemente

Das Alleinstellungsmerkmal unserer Anwendung ist es, viele verschiedene Module zu vereinen. Außerdem haben wir eine sehr breit gefächerte Zielgruppe, durch die wir Personen mit den unterschiedlichsten Krankheiten ansprechen wollen. Diese Diversität wollen wir mit unserem Branding ausstrahlen. Es besteht aus verschiedenen Formen und Farben, die Teil unserer Anwendung sind.

4.2.5. Design System

Icons, Buttons und klickbare Flächen

Da wir ein 8 Pixel-Grid verwenden, sind alle Icons 8, 16, 24 oder 32 Pixel groß. Unsere Icons sind, passend zum Rest der Anwendung, in einem geometrischen und reduzierten Look. Damit die Icons auch gut zu unserer Schriftart passen, runden wir die Linien an den Ecken und Enden nicht ab.

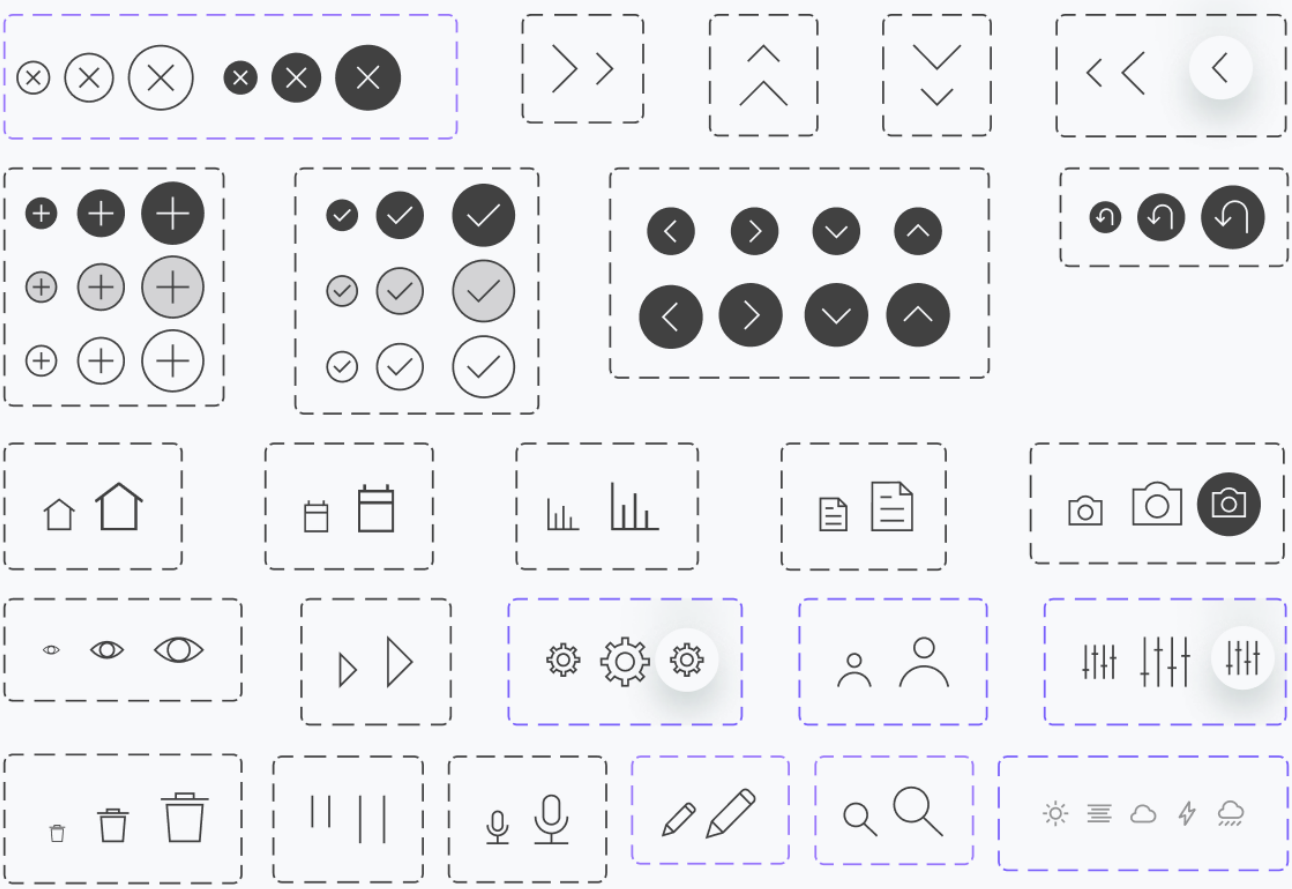


Abb. 9: Icons

Außerdem erhält jedes Modul ein eigenes Icon. Dadurch wollen wir gewährleisten, dass man schnellstmöglich und ohne den Text zu lesen, ein Modul erkennt.

Abb. 10: Icons der Module

Je nachdem, in welchem Modul man sich befindet, werden Buttons mit der Modul Farbe eingefärbt. Ob man einen Button mit oder ohne Inhalt verwendet, hängt vom Kontext ab. Sieht man in der Anwendung eine vollflächig bunte Fläche, bedeutet es, dass diese klickbar ist. So auch bei den Buttons. Für jeden Button gibt es vier States: enabled, hover, active und disabled.

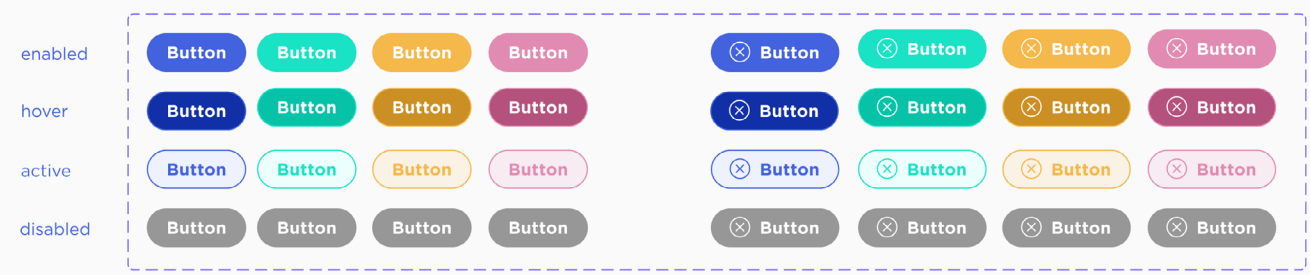


Abb. 11: Buttons in verschiednen States

Menü und Swipe-Funktion - Navigation innerhalb der App

Im Menü befinden sich vier Icons bzw. Buttons. Auf der rechten Seite liegt der “Hinzufügen”-Button. Klickt man darauf, gelangt man zur Detaileingabe. Er ist ganz rechts angeordnet, da die meisten User das Smartphone mit der rechten Hand bedienen. Dies führt dazu, dass der Abstand von Daumen zu Button am geringsten ist. Da wir davon ausgehen, dass dieser Button sehr oft getätigt wird, trägt dies zu einer positiven Benutzererfahrung bei. Wenn ein Icon ausgewählt ist, wird es in unserer Branding-Farbe eingefärbt. Damit auch Personen mit Probleme beim Unterscheiden von Farben die Auswahl erkennen können, gibt es zusätzlich einen kleinen Balken über dem ausgewählten Icon. Die Reihenfolge der Icons ist nach Häufigkeit der Nutzung von rechts nach links angeordnet.

Schiebt man im Dashboard ein Modul nach links, kommt man automatisch zur Detaileingabe dieses Moduls. Wischt man nach rechts, öffnet sich der Kalender. Allerdings wird nur das Modul als gefiltert angezeigt, welches man im Dashboard nach rechts gewischt hat.

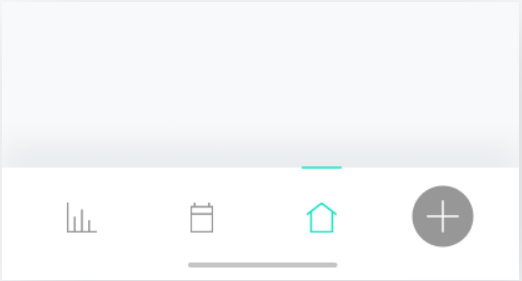


Abb. 12: Menü

Abb. 13: Swipe-Funktion Bearbeitung

Abb. 14: Swipe-Funktion Kalender

Slider - Bewertung von Einträgen

Da man bei jedem Eintrag die Wirkung, Intensität, Verträglichkeit oder Qualität bewerten kann, haben wir uns bei der Bewertung für einen Slider entschieden. Dabei kann man zwischen fünf Stufen auswählen. Wir haben uns bewusst für diese Anzahl an Stufen entschieden, da der User somit die Möglichkeit hat, auch eine neutrale Bewertung abzugeben, also Stufe drei. Im Hintergrund des Sliders befindet sich immer ein Text, der anzeigt, was man bewerten kann. Bei der default Einstellung ist der Text immer ganz sichtbar. Dadurch können wir gewährleisten, dass man die Beschreibung immer lesen kann.

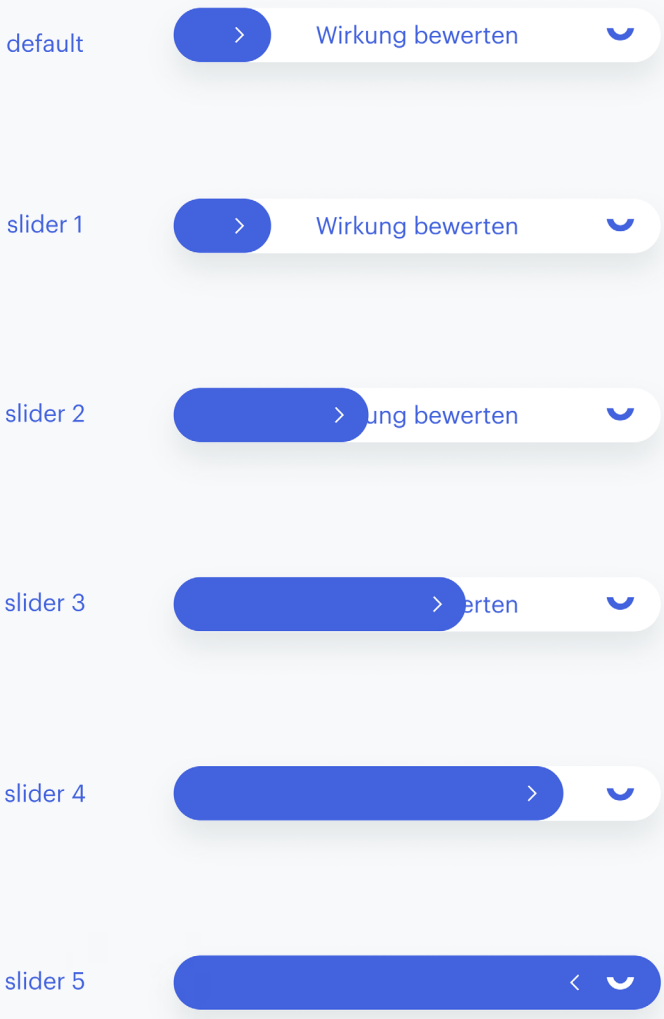


Abb. 15: Slider in Isolation

Abb. 16: Slider im Modul Bewegung

Figur und Regler - Schnelle Symptomeingabe

In der Detaileingabe der Symptome hat man die Möglichkeit eine Symptom-Kategorie auszuwählen, oder direkt eine neue zu Erstellen. Im Anschluss kann man bestimmen, wo man das Symptom am Körper spürt. Damit wir die Eingabe von Symptomen möglichst schnell gestalten, haben wir den Regler entwickelt. Dies ermöglicht die gleichzeitige Eingabe von Ort und Intensität. Zuerst klickt man mit dem Finger auf die Stelle am Körper, an der man das Symptom spürt. Im selben Schritt zieht man den Regler nach oben, so weit, bis man die gewünschte Intensität erreicht hat. Wenn man den Regler wieder loslässt, erscheint an der Stelle der Eingabe ein Kreis mit einer Zahl, die die Intensität anzeigt. Klickt man auf diesen Kreis, kann man noch mehr Informationen zur Detaileingabe hinzufügen. Je nachdem, welches Geschlecht man bei der Registrierung ausgewählt hat, erscheint die Figur im passenden Geschlecht. Durch einen kleinen Button rechts unten kann man den Körper drehen und auch auf der Rückseite Einträge machen.

Abb. 17: Regler in Isolation

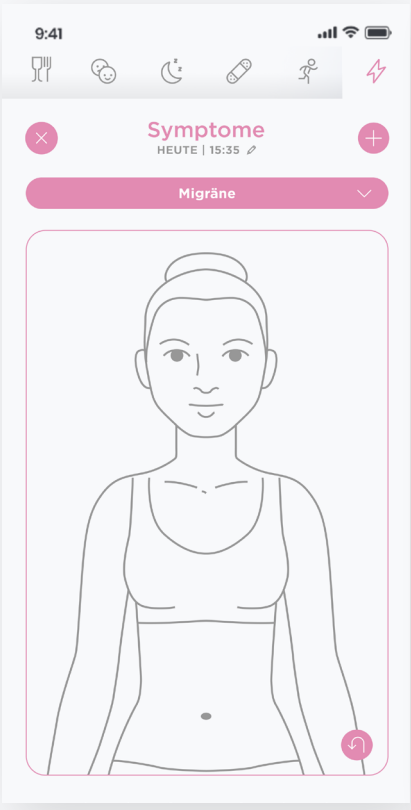


Abb. 18: Figur

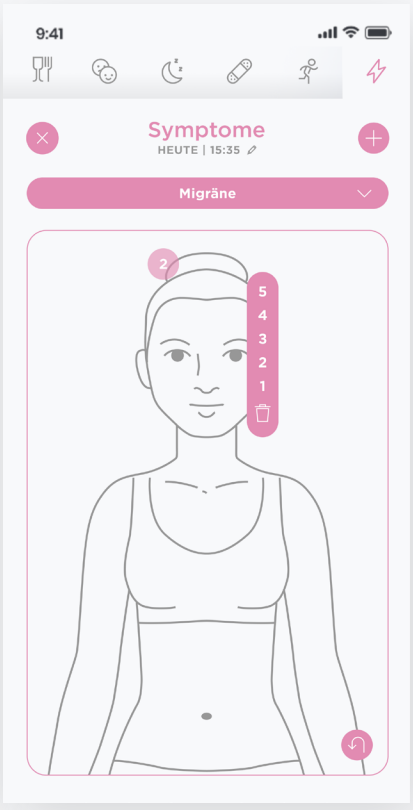


Abb. 19: Regler auf Figur

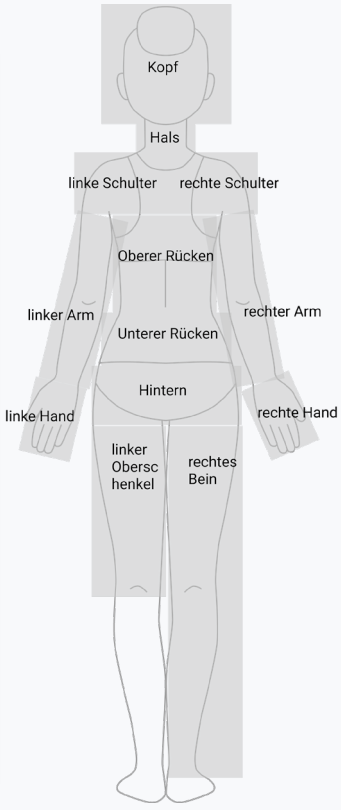


Abb. 20: Aufteilung der Figur

Modul-Menü - Navigation in der Detaileingabe

Abb. 21: Modul Menü

Um in der Detaileingabe zwischen verschiedenen Modulen navigieren zu können, gibt es ein Modul-Menü. Je nachdem, wie viele Module man eingeschaltet hat, erscheinen die entsprechenden Icons nebeneinander. Wenn ein Modul ausgewählt wurde, wird es mit einem helleren Hintergrund hinterlegt und in der Modul-Farbe eingefärbt. Wir haben uns bewusst für diese Form eines Klickbaren Menüs entschieden, da es die größte Übersichtlichkeit bietet und man sehr einfach und schnell, mit nur einem Klick, zwischen den Modulen navigieren kann. Ein Drehrad hätte beispielsweise nicht alle Module gleichzeitig angezeigt und man benötigt mehr Klicks um an das gewünschte Modul zu gelangen.

Drehrad und Filter - Navigation im Kalender

Damit der User schnell im Kalender zwischen verschiedenen Jahren und Monaten navigieren kann, haben wir zwei Drehräder eingebaut. Mit dem oberen kann man das Jahr bestimmen und mit dem unteren den Monat. Je länger man dreht, desto schneller bewegt sich das Drehrad. Außerdem bewegt sich der Kalender automatisch mit dem Drehen der Räder, sodass man immer den ausgewählten Monat, oder das ausgewählte Jahr sieht. Um sich bestimmte Module im Kalender anzeigen zu lassen, kann man die Filterung betätigen. Dort wird jedes Modul angezeigt, dass man in den Einstellungen ausgewählt hat. Durch die Checkboxen ist eine Mehrfachauswahl möglich.

Abb. 22: Kalender

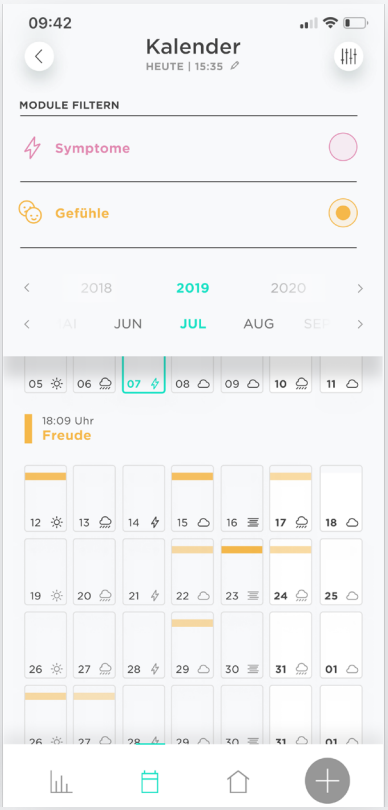


Abb. 23: Filterung Kalender

Checkboxen und Auswahl

Manchmal soll der User mehrere Möglichkeiten gleichzeitig auswählen können. Beispielsweise bei der Filterung von Modulen im Kalender. In diesem Fall verwenden wir Checkboxen. Die vollflächig farbige Punkte weisen darauf hin, dass man damit interagieren kann. Andererseits kann es sein, dass nur eine Kategorie zur Auswahl sinnvoll ist, z.B. bei der Wahl einer Symptomkategorie in der Detaileingabe. Hierfür werden gewöhnliche Buttons mit Icons verwendet.

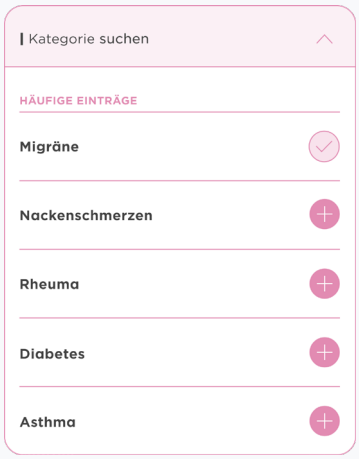


Abb. 24: Auswahl

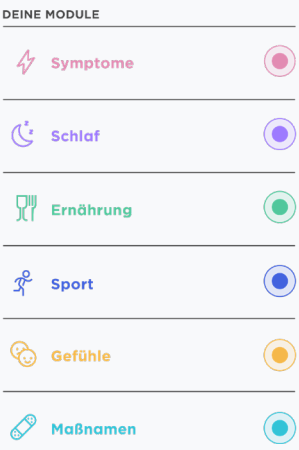


Abb. 25: Checkboxen

Eingabe und Suche

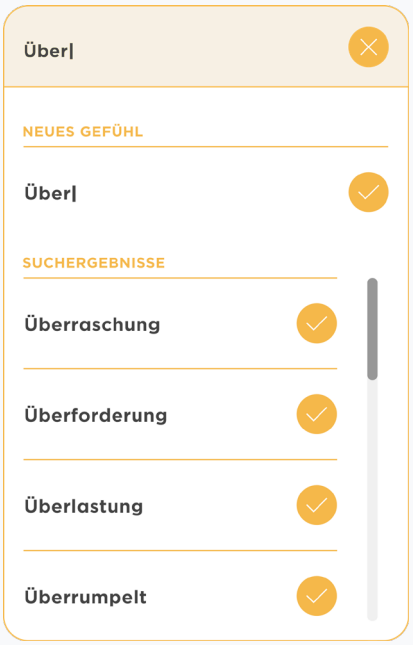


Abb. 26: Eingabe und Suche

Abb. 27: Eingabefeld vor erstem EIntrag

Eines unserer Alleinstellungsmerkmale ist die eigenständige Konfigurierbarkeit von Eingaben. In der Schnelleingabe im Dashboard kann der Nutzer z. B. ein Gefühl hinzufügen, indem er auf den Button “Gefühl hinzufügen” klickt. Dadurch werdem ihm die die häufigsten fünf Einträge angezeigt. Er hat aber auch die Möglichkeit nach einer Eingabe zu suchen, indem er das Wort oben im Textfeld eingibt. Dadurch kann er sich entweder einen eigenen neuen Eintrag generieren oder es werden ihm Vorschläge angezeigt, die zu seiner Eingabe passen könnten. Ab einer gewissen Länge, klappt sich die Eingabe nicht mehr aus, sondern wird scrollbar. Dadurch spart sich der Nutzer Zeit, wenn er wieder zurück zum Text-Eingabefeld gelangen will. Falls der User noch keine Eingaben gemacht hat, wird er darauf hingewiesen.

4.3. UX-WRITING

Um auch sprachlich einen einheitlichen Eindruck zu erwecken, haben wir einige Regeln festgelegt. Damit wollen wir erreichen, Texte und Beschreibungen so zu gestalten, dass sie wie von selbst für eine positive Nutzererfahrung sorgen und möglichst intuitiv sind. Nur der Inhalt des Textes soll einen Eindruck hinterlassen, nicht die Formulierung.

PERSÖNLICHE ANSPRACHE

Don't	**Wie geht es Ihnen?**
Do	**Wie geht es dir?**

AKTIONSVERWENEB VERWENDEN

Don't	**“Okay”, “Nein”**
Do	**„Verbinden“, „Senden“, „Abonnieren“, “Exportieren”, “Speichern”**

SPRACHEINGABE DURCH FRAGEN

Don't	**Bestimme den Ort deiner Migräne.**
Do	**Wo hast du Migräne?**

IM AKTIV SCHREIBEN

Don't	**“Suchen” sollte angeklickt werden, wenn Sie bereit sind, nach einem Artikel zu suchen.**
Do	**Klicke auf “Suchen”, um nach einem Artikel zu suchen.**

MIT DEM ZIEL BEGINNEN

Don't	**Tippe auf das Element, um seine Eigenschaften anzuzeigen**
Do	**Um die Eigenschaften des Elements anzuzeigen, tippe darauf**

DATUM SCHNELL KOMMUNIZIEREN

Don't	**08.07.2021**
Do	**“heute”, “gestern”**

IM PRÄSENS SCHREIBEN

Don't	**Video wurde heruntergeladen**
Do	**Video heruntergeladen**

ZIFFERN VERWENDEN

Don't	**Du hast zwei verpasste Anrufe**
Do	**Du hast 2 verpasste Anrufe**

BUTTONS EINHEITLICH UND KURZ BENENNEN

Don't	**“Neues Foto hinzufügen”, “zurück”, “Eingabe bestätigen”**
Do	**“Foto hinzufügen”, “Zurück”, “Bestätigen”**

ÜBERSCHRIFTEN EINHEITLICH UND KURZ BENENNEN

Don't	**“Deine häufigsten Einträge”, “Lebensmittel hier eingeben”**
Do	**“Häufige Einträge”, “Lebensmittel suchen”**

4.4. SCREENDESIGNS

4.4.1. Smartphone

Login, Register, Datenschutz, Einstellungen

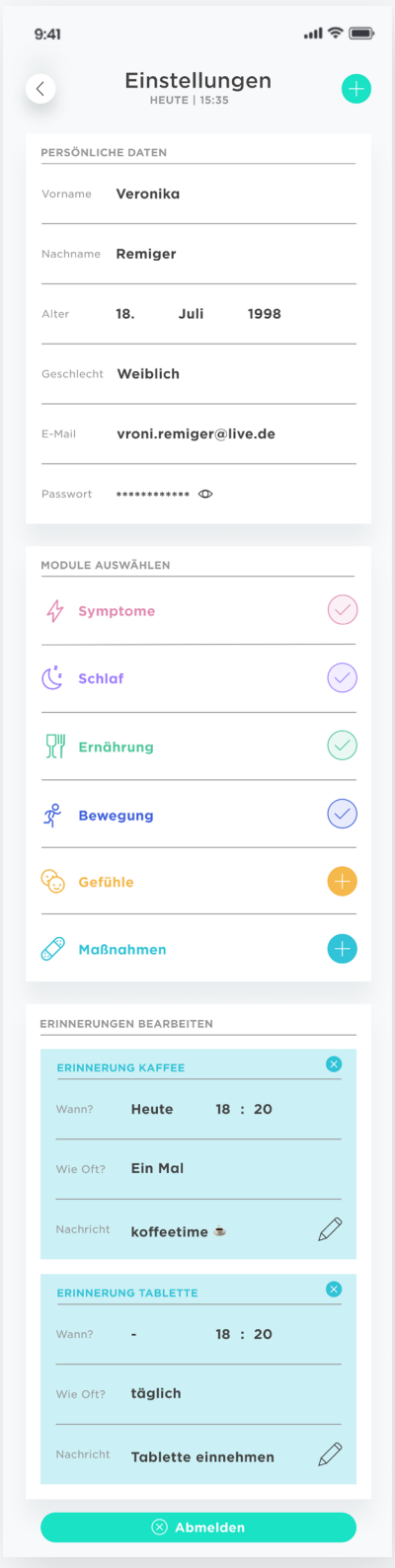


Abb. 28: Einstellungen

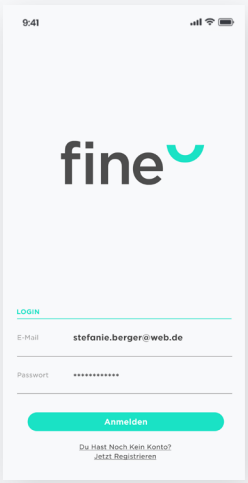


Abb. 29: Login

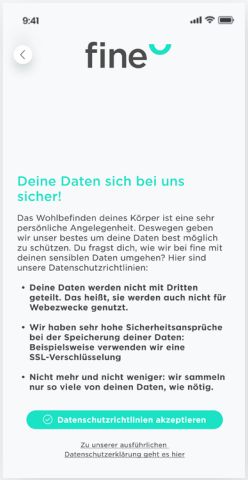


Abb. 30: Datenschutz

Abb. 31: Register

Da bei den Seiten Login, Register, Datenschutz und den Einstellungen die Funktion im Vordergrund steht, haben wir die Screens möglichst reduziert gestaltet. Wichtige Elemente und Buttons, oder Überschriften werden mit unserer Branding-Farbe hervorgehoben. Bevor man sich zum ersten mal registrieren will, wird man auf eine Datenschutz-Seite weitergeleitet. Da unsere Nutzer sensible Daten angeben, ist es bei unserer Anwendung besonders wichtig, Vertrauen aufzubauen. Deswegen erklären wir kurz und verständlich, welche Maßnahmen wir ergreifen, um die Daten unserer Nutzer zu schützen.
In den Einstellungen kann man nachträglich seine Angaben bearbeiten und auswählen, welche Module man tracken will. Außerdem werden hier alle Erinnerungen angezeigt, die man sich im Modul Maßnahmen generiert hat. Dabei kann man sich selbst einstellen, wann und wie regelmäßig man erinnert werden will. Außerdem kann man sich den Text der Erinnerung selber festlegen. So wollen wir gewährleisten, dass der Inhalt der Nachricht an die eigenen Bedürfnisse angepasst ist.

Einführungstutorial

Nach der Registrierung haben unsere User die Möglichkeit sich unsere Anwendung durch ein Einführungstutorial erklären zu lassen. Indem irrelevante Elemente verschwommen dargestellt werden, wird der Fokus auf den Bereich gelenkt, der gerade erklärt wird. Unsere Branding-Farbe kennzeichnet zusätzlich Fokuspunkte. Jederzeit kann man das Tutorial abbrechen, einen Schritt zurück, oder weiter gehen. Der letzte Schritt des Tutorials kann allerdings nicht übersprungen werden. Hier wählt man aus, welche Module man tracken will. Dies ist für jeden User relevant.

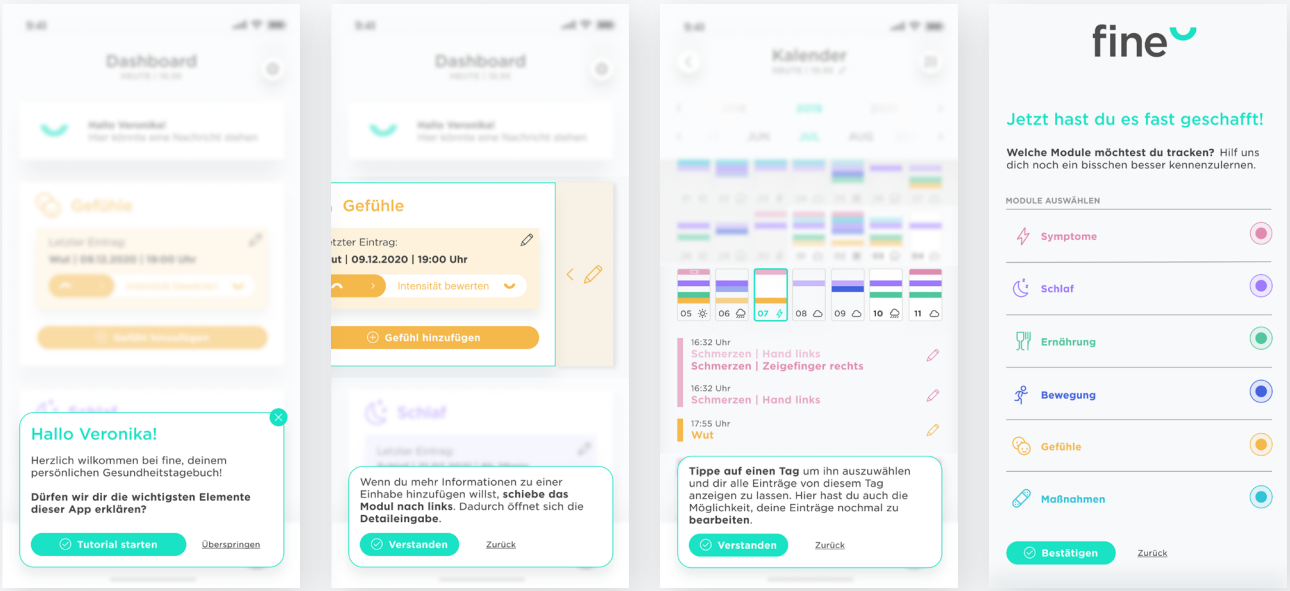


Abb. 32-35: Einführungstutorial

Dashboard

Unter der Überschrift befindet sich ein Smart-Stapel. Dort werden automatisch zur Tageszeit relevante Hinweise angezeigt. Wenn das System feststellt, dass ein User regelmäßig immer um 22:00 schlafen geht, könnte zur dieser Zeit ein Smart-Stapel ein Hinweis stehen, dass man seine Schlaf-Eingabe nicht vergessen soll. Man kann zwischen den verschiedenen Ebenen des Stapels navigieren, indem man die Karten nach oben oder unten wischt.

Außerdem findet man im Dashboard eine Übersicht aller Module, die man ausgewählt hat. Die Module dienen als Schnelleingabe. Über den Eingabebutton und unser intelligentes System, kann man möglichst schnell einen neuen Eintrag hinzufügen (siehe Designsystem: Eingabe und Suche). Zudem kann man durch den Regler seinen letzten Eintrag bewerten. Manche Nutzer wollen aber mehr Informationen zu einem Eintrag hinzufügen. Dafür haben wir die Detaileingabe entwickelt (siehe. Screendesigns - Smartphone - Module). Durch die Swipe-Funktion der Module im Dashboard (siehe Designsystem - Menü und Swipe-Funktion) gelangt man durch eine einzige Geste zur gewünschten Detaileingabe oder zum automatisch gefilterten Kalender.

Abb. 36: Dashboard

Kalender

Im Kalender hat man eine guter Übersicht über alle Einträge, die man in der Vergangenheit gemacht hat. Wenn man sich nur bestimmte Module anzeigen lassen will, kann man sich diese in der Filterung auswählen. Indem man nach oben und unten scrollt, kann man zwischen den Monaten wechseln. Zusätzlich, gibt es die Drehräder, die die Navigation zwischen großen Zeitabständen vereinfachen sollen. Im Kalender zeigen die Farben an, welche Module eingetragen wurden und durch die Farbabstufungen die Bewertung der Intensität. Klickt man auf einen Tag, werden alle Einträge angezeigt. Außerdem kann man dort Einträge bearbeiten, oder auch löschen, indem man einen Eintrag nach Links zieht.

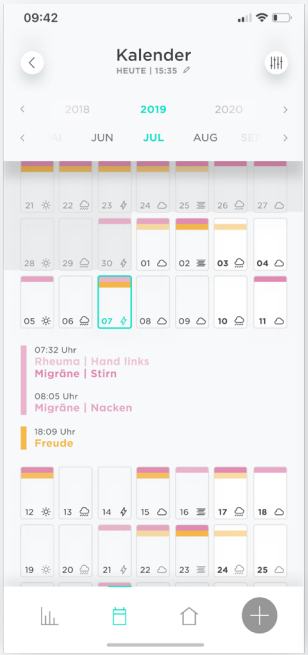


Abb. 37: Kalender

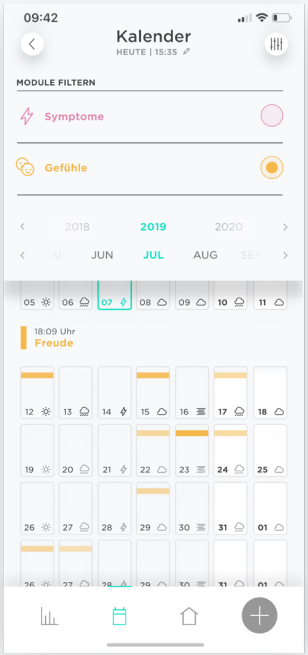


Abb. 38: Kalender FIlterung

Module

Die meisten Module sind nach demselben Prinzip aufgebaut. Folgende Punkte lassen sich bei der Dateneingabe hinzufügen:

- Mit dem Slider kann man die Intensität bewerten
- Durch das Eingabefeld kann man die Art Eintrags festlegen (Gefühl: Wut)
- Durch ein weiteres Eingabefeld kann man Schlagworte hinzufügen
- Im Notizfeld kann man eine Textnachricht, Sprachaufnahme, oder ein Foto hinzufügen

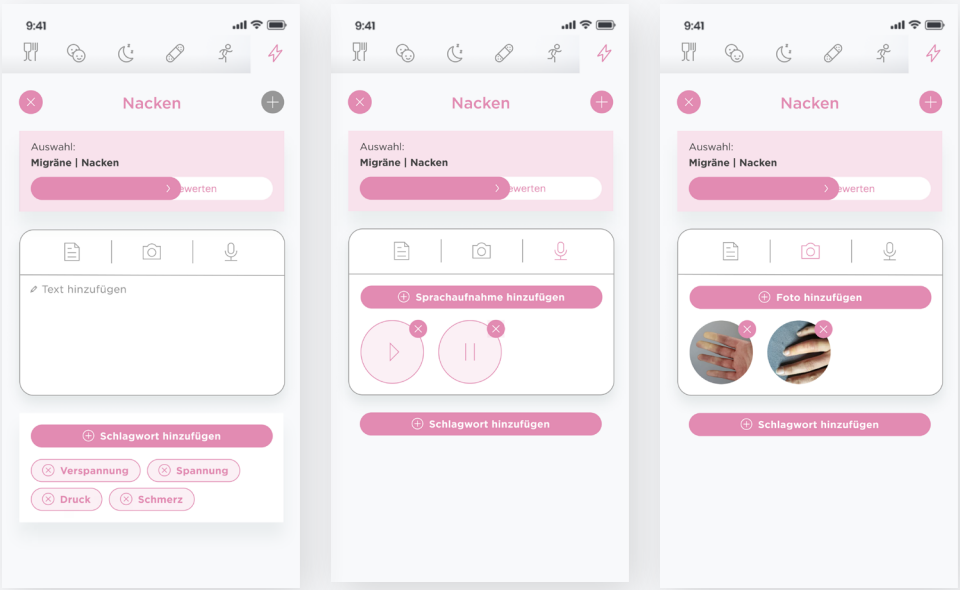


Abb. 39-41: Detaileingabe Symptome

Allerdings gibt es bei vielen Modul eine kleine Besonderheit:

- Symptome: Eingabe des Ortes des Symptoms durch die Figur
- Maßnahmen: Erinnerungen hinzufügen
- Ernährung: Eingabe der Lebensmittel durch automatische Foto-Erkennung
- Schlaf & Bewegung: Festlegung des Start- und Endzeit

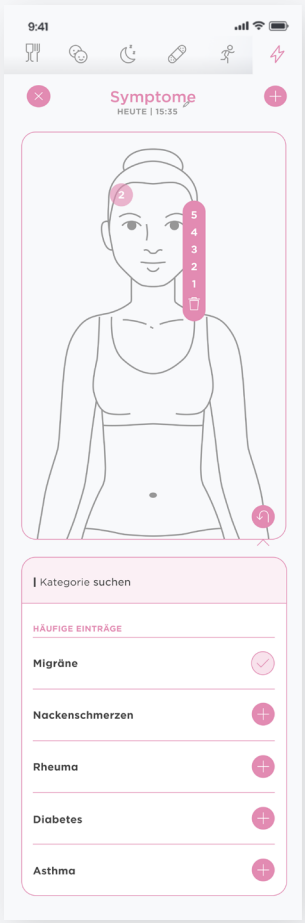


Abb. 42: Eingabe durch Figur

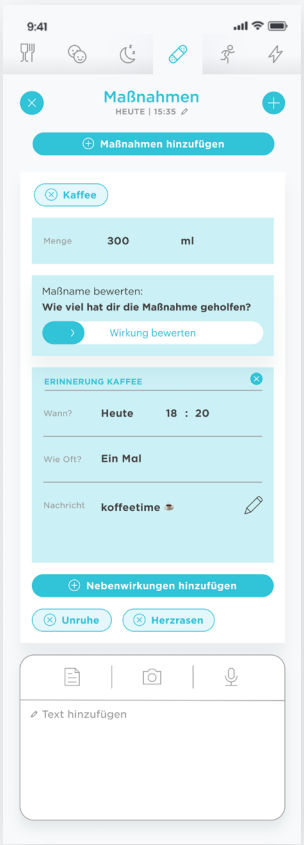


Abb. 43: Eingabe von Erinnerungen

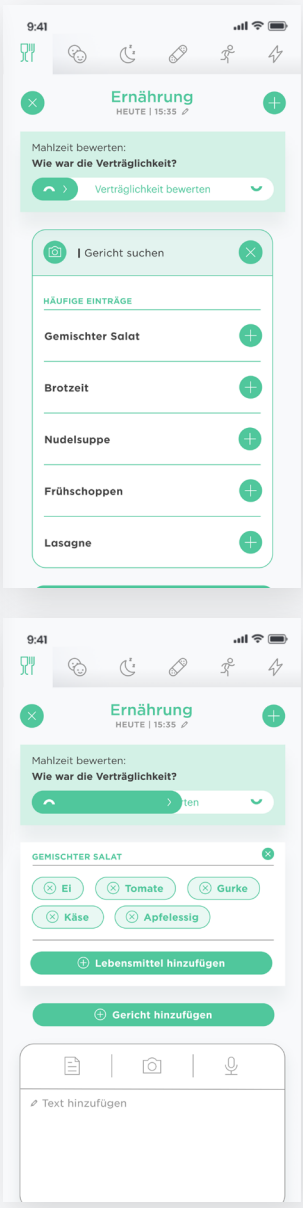


Abb. 44-45: Eingabe durch Kamera

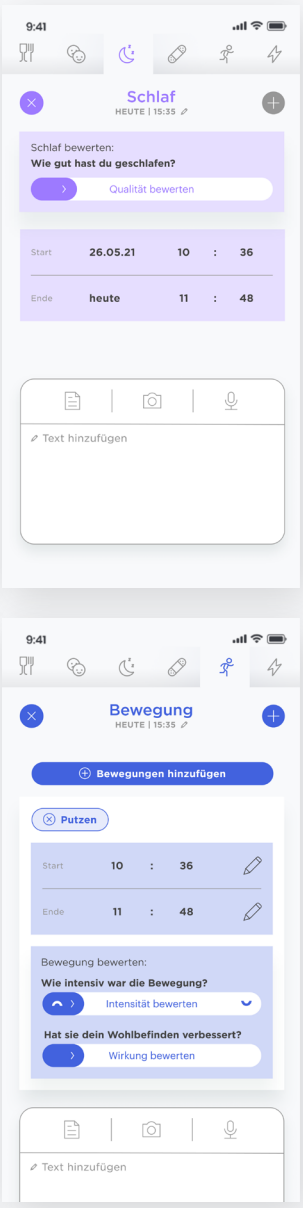
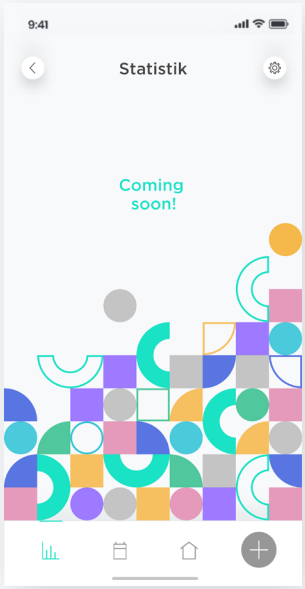


Abb. 46-47: Eingabe von Start- und Endzeit

Statistik



Leider haben wir es nicht mehr geschafft ein Konzept für die Statistik auszuarbeiten. Deswegen findet man an dieser Stelle einen „Coming soon“-Screen, passend im Design unseres Styleguides.

Abb. 48: Screendesign Statistik

4.4.2. Tablet

Da die meisten Personen Rechtshänder sind, würden sie mit ihrer Hand den kompletten Screen auf dem Tablet verdecken, wenn sie im Menü navigieren wollen. Deswegen haben wir das Menü auf die rechte Seite verlagert. Inhaltlich unterscheidet sich das Dashboard nicht von der mobilen Ansicht. Allerdings werden gleichzeitig die Module links und die Detaileingabe rechts angezeigt. So sieht man, wie sich bei der Eingabe gleichzeitig der Inhalt im Dashboard verändert. Im Kalender wird auf der rechten Seite dauerhaft die Filter-Option angezeigt. Im Login und Register ist genügend Platz auf der Tablet-Ansicht, um unser Branding zu integrieren. Dies steigert unseren Wiedererkennungswert.

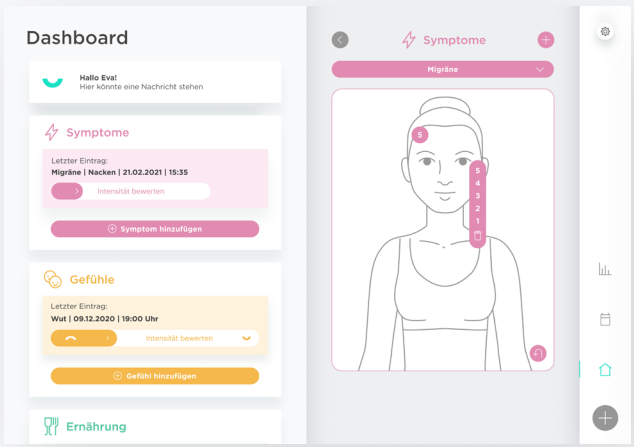


Abb. 49: Screendesign Tablet Dashboard und Symptome

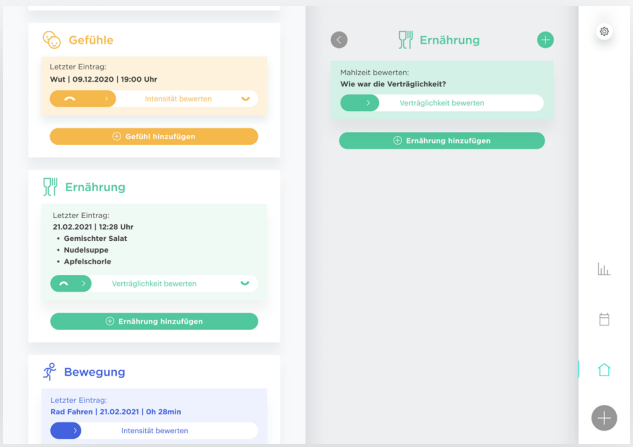


Abb. 50: Screendesign Tablet Dashboard und Ernährung

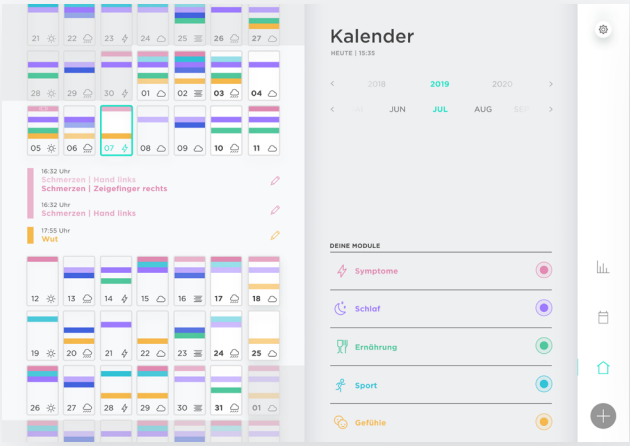


Abb. 51: Screendesign Tablet Kalender

Abb. 52: Screendesign Tablet Login

4.4.4. Marke

Video

Um unsere App zu vermarkten, haben wir ein Produktvideo gestaltet. Im Intro zeigen wir die Notwendigkeit für unsere App und unser Alleinstellungsmerkmal. Wir sind eine All-in-One-Web-App für die Verbesserung des Wohlbefinden unserer Nutzer.

Der Hauptteil zeigt die Komplexität unserer App. Diesen Teil kann man auch als Tutorial verstehen, der unserer Zielgruppe einerseits die Möglichkeiten unserer App zeigt, aber auch erklärt, wie sie funktionieren.

Den Schluss haben wir möglichst kurz gehalten. Hier zeigen wir, dass es auch eine Tablet-Ansicht gibt und wiederholen unseren Slogan.

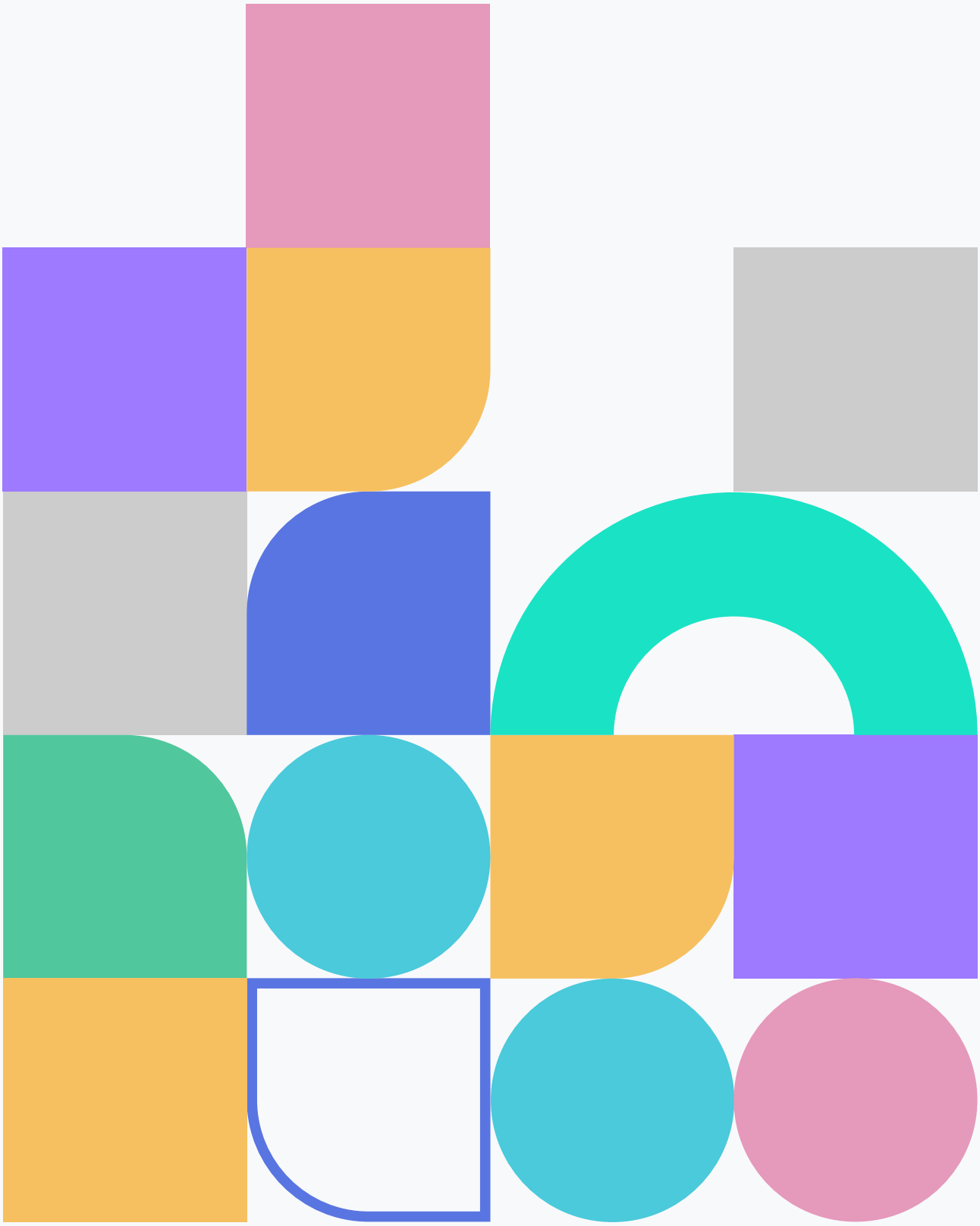
Abb. 53: Produktvideo

App-Icon

Weil der lächelnde Mund das höchste Wiedererkennungsmerkmal im Logo ist, wird er beim App-Icon erneut eingesetzt. Anders als beim Logo ist die Form in Weiß eingefärbt und liegt auf einem flächig türkisem Hintergrund. Dadurch hat das App-Icon einen größeren Farbanteil und lässt sich einfacher von anderen unterscheiden. Durch die auffällige Farbe hebt es sich auch gut von anderen ab. Damit das Icon ausgeglichen wirkt, liegt die Form nicht zentriert im Hintergrund, sondern befindet sich in der optischen Mitte.

Abb. 54: App-Icon fine

5. ENTWICKLUNGS-UMGEBUNG



Für einen reibungslosen Projektstart und eine gute Zusammenarbeit der Entwickler war es sehr wichtig, direkt zu Beginn des Produktionssemesters eine funktionierende Entwicklungsumgebung zu haben. Ein Problem war, dass wir in der Gruppe unterschiedliche Betriebssysteme (MacOS und Windows) nutzen. Deshalb war es wichtig, eine Möglichkeit zu finden, wie wir trotzdem problemlos zusammenarbeiten können.

Benjamin hat deshalb bereits in den Semesterferien damit begonnen, die Entwicklungsumgebung aufzusetzen und einen ersten funktionierenden Prototyp zu erstellen.

5.1. GITLAB

Abb. 55: Logo GitLab

Für die Versionsverwaltung unserer Anwendung haben wir GitLab verwendet. Mithilfe von Branches ermöglicht es, mehreren Entwicklern gleichzeitig an einem Projekt zu arbeiten und Änderungen anschließend wieder zusammenzufügen. Außerdem bietet GitLab viele weitere hilfreiche Tools für die Softwareentwicklung wie z. B. Continuous Integration oder Out-of-the-Box Pipelines. Diese haben wir jedoch nicht verwendet.[1]

Unser Projekt haben wir in ein Frontend- und ein Backend-Repository unterteilt. Dies hat einige Vorteile. Die unabhängige Versionierung der Repositories ist dabei einer der wichtigsten Punkte. Außerdem kann parallel am Backend und Frontend gearbeitet werden. Damit dies möglich ist, haben die Frontend-Entwickler mit Mockup-Daten gearbeitet, solange die API-Endpoints noch nicht zur Verfügung standen. So konnte bereits relativ früh der Look and Feel der Anwendung getestet und anschließend mit der API verknüpft werden.

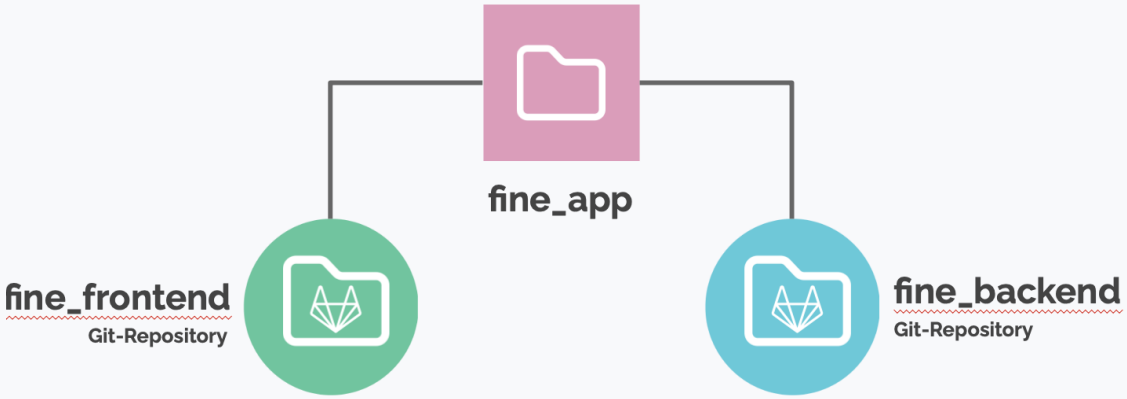


Abb. 56: Versionsverwaltung

Alle Aufgaben haben wir mithilfe von Jira verwaltet. Für jedes Ticket in Jira wurde ein eigener Branch mit der entsprechenden Ticketnummer angelegt. Sobald ein Ticket erledigt wurde, hat der zuständige Entwickler den Branch in das entsprechende Repository gepusht und einen Merge-Request für den Master erstellt. Bei vielen Branches und Merge-Requests kann das Ganze sehr schnell unübersichtlich werden. Deshalb war bei uns Sarah als Projektmanager für das Mergen zuständig. So hatten wir immer eine Person, welche den Überblick über den aktuellen Stand im Master hat. Sobald der Master geupdated wurde, hat Sarah alle Entwickler per Slack benachrichtigt. Diese haben dann den aktuellen Stand des Masters in ihre Branches gemerged.

5.2. DOCKER

Abb. 57: Logo Docker

Wie bereits in der Dokumentation des Konzeptionssemesters erwähnt, haben wir uns dazu entschlossen, Docker zu verwenden. Docker ermöglicht es, den Code der Anwendung und all seine Abhängigkeiten in einen Container zu verpacken. Dieser ist ein eigenständig ausführbares Softwarepaket, das alles bereitstellt, was zum Ausführen der Applikation nötig ist.[2]

Für die Verwendung von Docker gab es viele Gründe. Der größte Vorteil war, dass Docker die Entwicklung auf unterschiedlichen Betriebssystemen sehr einfach macht. Es werden automatisch alle benötigten Images und Abhängigkeiten unseres Projekts installiert. So ist sichergestellt, dass es zu keinen Problemen wegen unterschiedlicher Versionen von Images oder Abhängigkeiten kommt. Außerdem ermöglicht Docker auch nicht Informatik-affinen Gruppenmitgliedern ein schnelles Aufsetzen der Entwicklungsumgebung.

Auch der Livegang ist durch Docker einfacher von der Hand gegangen. Mehr dazu im Kapitel “Livegang”.

Docker Compose ist ein Tool, welches zum Erstellen von Multi-Container-Anwendungen entwickelt wurde. Der große Vorteil gegenüber normalen Docker-Containern ist, dass man mit einem einzigen Befehl mehrere Container starten und weitere Einstellungen vornehmen kann. So kann zum Beispiel festgelegt werden, ob Daten aus den Containern lokal auf dem Rechner gespeichert werden sollen. Docker Compose findet beim erneuten Starten der Container die verknüpften Ordner und stellt den alten Stand wieder her. Außerdem erkennt Docker Compose, ob die Anwendung aktualisiert wurde und erstellt nur in diesem Fall einen neuen Container.[3]

Da unsere Anwendung in drei Docker-Containern läuft, lag es nahe Docker Compose zu verwenden. Wir haben ein Docker Compose-File für das Frontend und das Backend. Im Gegensatz zur Verwendung von einem normalen Dockerfile benötigt man mit Docker Compose nur einen einfachen Befehl für die Konfiguration und den Start der Container.

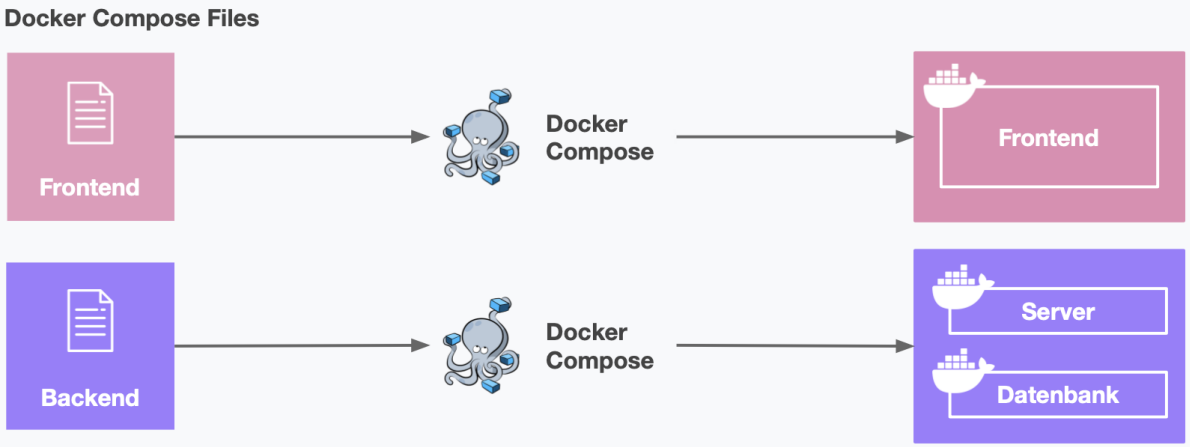


Abb. 58: Docker Compose

Im Backend haben wir einen Container mit einem Node.js-Server und einen Container mit einer MongoDB. Beide Services werden über ein Docker Compose-File verknüpft und gestartet. Im Folgenden werden wir dieses genauer erklären.

```
version: „3“
services:
 server:
   container_name: backend
   build:
       context: .
   depends_on:
     - mongo
   volumes:
   - „.:/backend“
   - „/backend/node_modules“
   restart: always
   ports:
   - „3000:3000“
   external_links:
   - mongo

 mongo:
   container_name: mongo
   image: mongo
   ports:
     - „27017:27017“
   environment:
     - MONGO_INITDB_DATABASE=fine_mongodb
     - MONGO_INITDB_ROOT_USERNAME=admin
     - MONGO_INITDB_ROOT_PASSWORD=test123
   volumes:
     - „./mongo-init.js:/docker-entrypoint-initdb.d/mongo-init.js:ro“
     - „./_mongodb_data:/data/db“
```

Quellcode. 1: Docker Compose-File Backend

Für den Node.js-Server werden im Compose-File die Ports und die Volumen festgelegt. Volumen ermöglichen es Docker, Daten lokal auf dem Filesystem des Nutzers zu speichern. Dies machen wir beispielsweise mit den Node-Modules. Weiterhin ermöglichen sie es, das lokale Projektverzeichnis auf unserem Computer innerhalb des Containers einzuhängen. Dies ermöglicht es, den Code im laufenden Betrieb zu ändern, ohne dass ein neues Image erstellt werden muss.

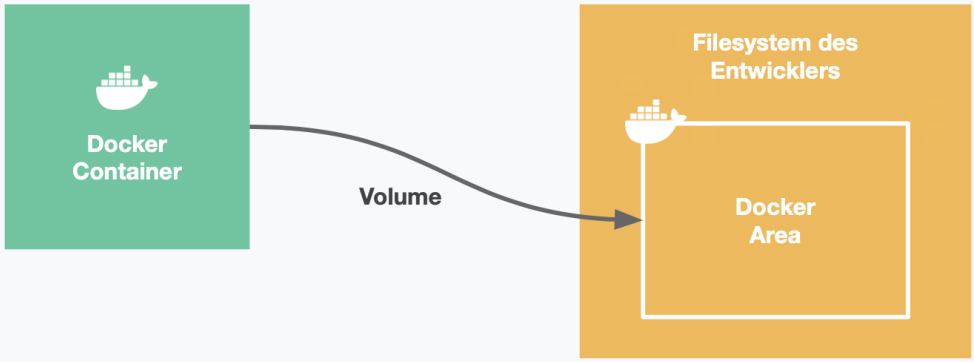


Abb. 59: Volumes in Docker Compose

Außerdem wird festgelegt, dass der Node.js-Server von der MongoDB abhängig ist. Dies ist wichtig, damit der Container der MongoDB als erstes gestartet wird.

Für die MongoDB werden Image, Ports und Pfade für die Volumen definiert und der Admin-Nutzer für die Datenbank angelegt. Dies funktioniert mithilfe von Environment-Variablen, welche den Datenbanknamen, Benutzernamen und Passwort enthalten. Es wird außerdem festgelegt, dass der aktuelle Stand der Datenbank lokal auf dem Filesystem des Nutzers gespeichert wird. So sind die Daten auch nach einem Neustart des Containers noch verfügbar. Auch ein Script zum Anlegen eines normalen Nutzers wird mithilfe von Volumen aus dem lokalen Projektverzeichnis in den Order “docker-entrypoint-initdb” kopiert. Alle enthaltenen Scripte in diesem Ordner werden beim Start des Containers vom MongoDB-Image automatisch ausgeführt.

```
version: „3“
services:
 frontend:
   container_name: frontend
   build:
     context: .
   volumes:
     - „.:/frontend“
     - „/frontend/node_modules“
   ports:
     - „8080:8080“
   environment:
     - CHOKIDAR_USEPOLLING=true
```

Quellcode 2: Docker Compose-File Frontend

Das Frontend besteht aus einem Container, in dem unsere Vue.js-Anwendung läuft. Theoretisch ist für diesen Container kein Compose-File notwendig. Wir haben uns trotzdem dafür entschieden, da so die Einstellungen für Hot-Reloads, die Volumen und die zu verwendenden Ports gesetzt werden können. Wie auch bereits beim Node.js-Server legen wir die Node-Modules im lokalen Projektverzeichnis des Nutzers ab und hängen dieses Verzeichnis innerhalb des Containers ein. Zum Schluss wird noch eine Environment-Variable gesetzt, damit Hot-Reloads möglich sind.

5.3. LINTER

Abb. 60: Logo ES-Lint

Ein Linter ist ein Tool zur Quellcodeanalyse. Er erkennt Probleme und kann viele davon auch automatisch beheben. Dazu gehören sowohl Programmierfehler, als auch stilistische Fehler. Für unser Projekt nutzen wir ESLint. Seit 2018 wird ESLint beim Erstellungsprozess von neuen Vue.js Projekten von den Entwicklern empfohlen.[4]

Wir verwenden ESLint im gesamten Projekt um Fehler zu erkennen, eine gleichbleibende Codequalität zu gewährleisten und ein einheitliches Erscheinungsbild zu wahren. Wir haben die Standards von AirBnB verwendet und geringfügig an unsere Bedürfnisse angepasst.

6. ARCHITEKTUR

Dieses Kapitel soll einen kurzen Überblick über die Architektur unserer Applikation geben, bevor in den nächsten beiden Kapiteln genauer auf das Back- und Frontend eingegangen wird.

6.1. BACKEND

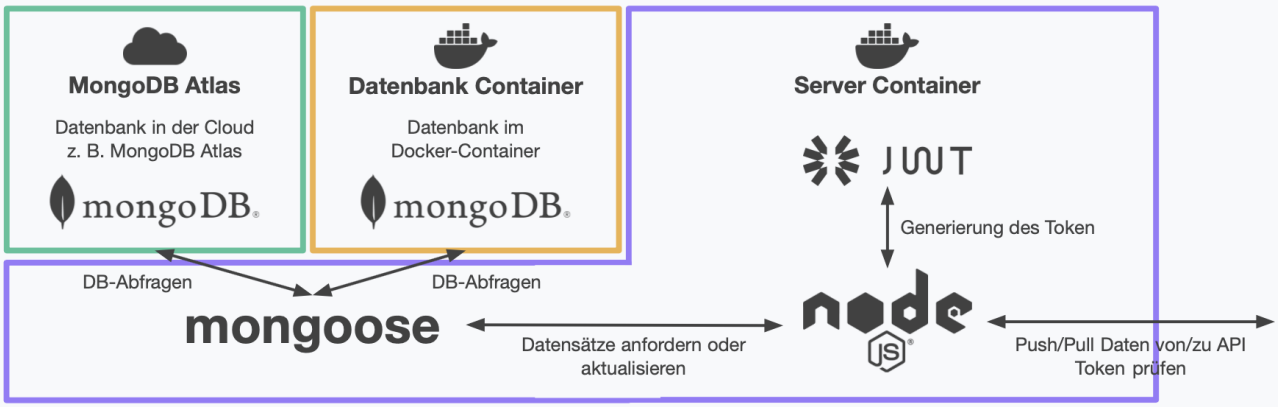


Abb. 61: Architektur Backend

Hier sehen wir den Aufbau unseres Backends. Wie bereits im letzten Kapitel erwähnt haben wir einen Container für die Datenbank und einen Container für den Server.

Als Datenbank kann entweder eine lokale MongoDB im Docker Container oder eine MongoDB, welche z. B. bei MongoDB Atlas gehostet wird, verwendet werden. Die Java-Script-Bibliothek Mongoose nutzen wir zum Daten abfragen, speichern, bearbeiten und löschen in der MongoDB.

Unser Express.js-Server stellt die entsprechenden API Endpoints zur Verfügung. Mit JSON Web Tokens (JWT) generieren wir Tokens, welche zur Authentifizierung der Nutzer dienen.

6.2. FRONTEND

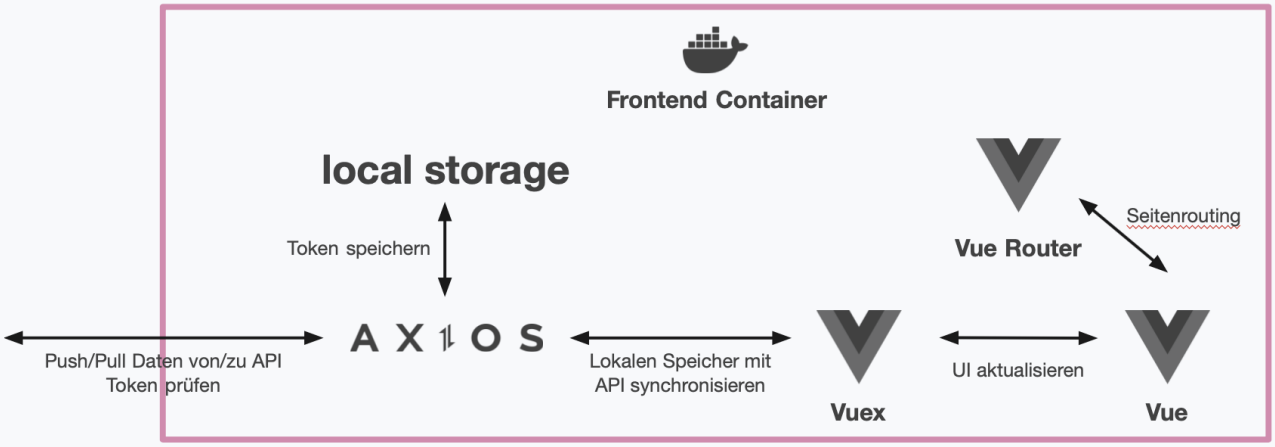


Abb. 62: Architektur Frontend

Im Frontend haben wir unsere Vue.js-Applikation. Da diese relativ komplex ist, haben wir uns bei der Verwaltung unserer Daten für den Einsatz von Vuex entschieden. So werden alle Daten an einem zentralen Ort verwaltet und können von allen Komponenten abgerufen werden. Axios nutzen wir um die Anfragen an unsere API zu stellen. Die Daten werden anschließend im Vuex-Store gespeichert.

Die JWT werden im Local Storage abgelegt, damit sie auch nach einem Reload der Seite vorhanden sind. Sie werden bei jeder Anfrage im Axios-Header an die API mitgesendet, um zu bestätigen, dass der Nutzer authentifiziert ist. Außerdem verwenden wir Vue Router um zwischen unseren verschiedenen Views zu wechseln.

7. BACKEND

7.1. RÜCKBLICK

In der Dokumentation des Konzeptionssemesters haben wir uns noch nicht abschließend auf einen Schnittstellen-Standard und eine Datenbank geeinigt.

Bei den Schnittstellen-Standards standen REST und GraphQL zur Auswahl. Wir haben uns für REST entschieden, da es eine unschlagbare Kompatibilität bietet. Nahezu alle APIs verfügen über eine REST-Schnittstelle. Falls wir z. B. eine der in der Dokumentation des Konzeptionssemesters genannten APIs zur Sprach- oder Bilderkennung nutzen wollen, müssten wir REST verwenden, da sie keine Schnittstelle für GraphQL bieten.

Bei den Datenbanken standen MongoDB und PostgreSQL zur Auswahl. Sie unterscheiden sich hauptsächlich durch das Datenbankmodell. So ist MongoDB eine dokumentenorientierte und PostgreSQL eine relationale Datenbank. Wir haben uns für MongoDB entschieden, da man zu Beginn kein festes Datenbankschema festlegen muss und so flexibel auf Änderungen bei den Anforderungen und Daten reagieren kann. Außerdem bietet MongoDB eine sehr große Community und wird oft mit den anderen Technologien aus unserem Tech-Stack verwendet.

Im Nachhinein war unsere Entscheidung richtig, da sich im Laufe des Projekts noch einige Änderungen an unserer Datenstruktur ergeben haben.

Beim Server setzen wir, wie bereits in der Dokumentation des Konzeptionssemesters festgelegt, auf Node.js und Express.js. Node.js haben wir verwendet, da wir so im gesamten Projekt dieselbe Programmiersprache (JavaScript) und das gleiche Datenformat (JSON) zum Austausch nutzen können. Auch die Vielzahl an Erweiterungen, eine sehr gute Dokumentation und eine große Community sprachen dafür.

7.2. DATENBANK

Wie bereits im letzten Abschnitt erwähnt, haben wir als Datenbank MongoDB gewählt. MongoDB ist eine dokumentenorientierte Datenbank. Dokumentenorientierte Datenbanken zählen zu den NoSQL-Datenbanken und verzichten auf einen relationalen Ansatz. Sie haben keine Tabellen, welche einem festen Datenbankschema unterliegen, sondern einzelne, eindeutig identifzierbare Dokumente. Sie sind besonders für unstrukturierte Daten gut geeignet.[5, 6]

Zur Modellierung unserer Daten haben wir uns für den Einsatz von Mongoose entschieden. Mongoose ist eine JavaScript Bibliothek für Object Data Modeling für MongoDB und Node.js.

Im ersten Schritt haben wir mithilfe von Mongoose Schemata für alle Daten angelegt. Jedes Modul (Emotionen, Symptome, Ernährung, Schlaf, Bewegung, Maßnahmen) hat sein eigenes Schema bekommen, da unterschiedliche Daten gespeichert werden müssen. In unserem Prototyp haben wir die Schemata für Emotionen und Symptome umgesetzt. Außerdem haben wir noch Schemata für den User und die Symptomkategorien angelegt. In den Schemas werden die Objekte, deren Datentyp und weitere Parameter festgelegt.

So sieht ein vereinfachtes Mongoose-Schema aus:

```
const Schema = mongoose.Schema({
 userid: {
   type: mongoose.Schema.Types.ObjectId,
   ref: „User“,
   required: true,
 },
 date: {
   type: Date,
   default: Date.now,
 },
 intensity: {
   type: Number,
   required: true,
 },
...
});

module.exports = mongoose.model(„Schema“, Schema);
```

Quellcode 3: Vereinfachtes Mongoose-Schema

Für jedes Schema wird in der Datenbank eine Kollektion erstellt, in der die sogenannten Dokumente abgelegt werden. Wie man beim Objekt userid sieht, können andere Dokumente aus der selben oder einer anderen Collection referenziert werden. In unserem Fall nutzen wir die Referenz, um die Moduleinträge einem User zuzuordnen.

Ein Dokument vom Schema Emotion in der Datenbank

_id: ObjectId(“60a61a2d773cf40a917db3f3”)
userid: ObjectId(“60bf58f09601e7282e0feb12”)
date:0233-12-31T23:06:32.000+00:00
intensity:”3”
title:“Wut“
photos: Array
audio: Array
tags: Array

Quellcode 4: Dokument Schema Emotions

Das gleiche System wird auch beim Nutzer verwendet. So hat der Nutzer Arrays, in denen seine Einträge z. B. für das Modul Emotions gespeichert werden.

Ein Dokument vom Schema User in der Datenbank

_id: ObjectId(“60bf58f09601e7282e0feb12”)
firstName:“Max“
lastName:“Mustermann“
email:“max@example.com“
password:“$2a$10$qmg5YBtoeebyZK3aBjvHU.5t04OYpn3LLpcUPzQzX3afA0n65coXm“
emotions: Array

```
0:ObjectId(“60e362e067e21f386f031d2b”)
1:ObjectId(“60e362e167e21f386f031d2d”)
2:ObjectId(“60e362e167e21f386f031d2e”)
```

Quellcode 5: Dokument Schema User

Mithilfe von Mongoose lässt sich sehr einfach abändern, welche Datenbank verknüpft werden soll. Für die Entwicklung haben wir meist eine Datenbank in der Cloud von MongoDB Atlas verwendet. Dies hatte den Vorteil, dass wir Änderungen in der Datenbank in Echtzeit auf einem schönen Userinterface sehen konnten. Die Verbindungen zur Datenbank haben wir in einem .env-File gespeichert, sodass wir nur eine Variable ändern mussten, um auf die lokale MongoDB im Dockercontainer umzuschalten.

7.3. SERVER & API

Abb. 63: Logo Node.js

Abb. 64: Logo Express

Abb. 65: Logo REST

Im Backend haben wir Node.js und Express.js verwendet. Node.js ist eine JavaScript-Laufzeitumgebung und ermöglicht die serverseitige Programmierung in JavaScript. Die Installation von Node.js beinhaltet den Paketmanager npm. Dieser erlaubt das einfache Hinzufügen von JavaScript-Modulen zum eigenen Projekt. Unter https://www.npmjs.com/ findet man eine Bibliothek mit einer großen Auswahl an Modulen, welche unterschiedlichste Funktionalitäten anbieten.[7, 8]

Express ist ein Webanwendungsframework, welches in Kombination mit Node.js meist als Serverframework zur Erstellung von Webanwendungen genutzt wird.[9]

Wir haben Express.js für die Erstellung des Webservers und der Rest API genutzt.
Bei REST APIs werden Daten mithilfe sogenannter Ressourcen adressiert. Um diese Ressourcen eindeutig identifizieren zu können, werden verschiedene Endpunkte mit unterschiedlichen URLs definiert. Diese Endpunkte können mithilfe von HTTP-Methoden angesprochen werden. Die REST API hat verschiedene Befehle. Dazu gehören unter anderem:

- GET (Ressource lesen)
- POST (Neue Ressource erstellen)
- PUT/PATCH (Ressource aktualisieren oder bearbeiten)
- DELETE (Ressource löschen)

Der Server entscheidet anschließend, welche Daten zurückgegeben werden. Für REST hat in unserem Fall vor allem die Standardisierung und die starke Verbreitung gesprochen.[10]

7.3.1. Routen

Wir haben folgende Routen für unsere Anwendung definiert:

Name	Route
Authentifizierung	/api/auth
Nutzer	/api/user
Symptomkategorien	/api/symptomCategories
Emotionen	/api/emotions
Symptome	/api/symptoms

Folgende Statuscodes können von der API zurückgegeben werden:

Status	Bedeutung	Befehl
200	OK	GET, PATCH, DELETE
201	Created	POST
401	Unauthorized	GET, PATCH, DELETE
404	Not found	GET, PATCH, DELETE
400	Bad request	POST, PATCH
500	Internal server error	

Authentifizierung

Diese Route wird zur Registrierung und zum Login des Nutzers angesprochen. Die beiden Endpunkte sind die einzigen in der Anwendung, welche ohne Autorisierung aufgerufen werden können.

Vor dem Anlegen des Nutzers wird sein Passwort mithilfe von der Bcrypt-Bibliothek gehashed, um die Sicherheit zu erhöhen.

Authentifizierung	Befehl	Endpunkt	Status
Nutzer registrieren	POST	/api/auth/register	201
Nutzer anmelden	POST	/api/auth/login	200

Nutzer

Diese Route wird angesprochen, wenn ein Nutzer geladen, aktualisiert oder gelöscht werden soll. Um diese Endpunkte anzusprechen, muss der Nutzer authentifiziert sein.

Nutzer	Befehl	Endpunkt	Status
Nutzer zurückgeben	GET	/api/user/:userId	200
Nutzer löschen	DELETE	/api/user/:userId	200
Nutzer aktualisieren	PATCH	/api/user/:userId	200
Ausgewählte Module des Nutzers aktualisieren	PATCH	/api/user/modulesSelected/:userid	200

Symptomkategorien

Diese Route wird angesprochen, wenn Symptomkategorien geladen, hinzugefügt, aktualisiert oder gelöscht werden soll. Um diese Endpunkte anzusprechen, muss der Nutzer authentifiziert sein.

Symptomkategorien	Befehl	Endpunkt	Status
Alle Symptomkategorien zurückgeben	GET	/api/symptomCategories	200
Symptomkategorie hinzufügen	POST	/api/symptomCategories	201
Symptomkategorie löschen	DELETE	/api/symptomCategories/:symptomCategoryId	200
Symptomkategorie aktualisieren	PATCH	/api/symptomCategories/:symptomCategoryId	200
Eine Symptomkategorie zurückgeben	GET	/api/symptomCategories/:symptomCategoryId	200^

Emotionen

Diese Route wird angesprochen, wenn Emotionen geladen, hinzugefügt, aktualisiert oder gelöscht werden soll. Um diese Endpunkte anzusprechen, muss der Nutzer authentifiziert sein.

Emotionen	Befehl	Endpunkt	Status
Alle Emotionen zurückgeben	GET	/api/emotions	200
Emotion hinzufügen	POST	/api/emotions	201

Emotion löschen	DELETE	/api/emotions/:emotionId	200
Emotion aktualisieren	PATCH	/api/emotions/:emotionId	200
Eine Emotion zurückgeben	GET	/api/emotions/:emotionId	200
Die zuletzt eingetragene Emotion zurückgeben	GET	/api/emotions/lastEmotionEntry	200

Symptome

Diese Route wird angesprochen, wenn Emotionen geladen, hinzugefügt, aktualisiert oder gelöscht werden soll. Um diese Endpunkte anzusprechen, muss der Nutzer authentifiziert sein.

Symptome	Befehl	Endpunkt	Status
Alle Symptome zurückgeben	GET	/api/symptoms	200
Symptom hinzufügen	POST	/api/symptoms	201
Symptom löschen	DELETE	/api/symptoms/:symptomId	200
Symptom aktualisieren	PATCH	/api/symptoms/:symptomId	200
Ein Symptom zurückgeben	GET	/api/symptoms/:symptomId	200
Das zuletzt eingetragene Symptom zurückgeben	GET	/api/symptoms/lastSymptomEntry	200

7.3.2. Validierung

Bei der Anmeldung und beim Login überprüfen wir zusätzlich mithilfe der Bibliothek Joi, ob die Eingaben des Nutzers den Vorgaben entsprechen. Joi ist eine Schema-Beschreibungssprache. Mit Joi erstellt man Schemata für JavaScript-Objekte, um die Eingabe zu validieren, bevor sie an die Datenbank weitergegeben wird.[11]

So kann man zum Beispiel validieren, ob eine gültige E-Mail eingegeben oder die Mindestlänge des Passworts eingehalten wurde.

Hier ein Beispiel der Validation für die Angaben aus der Register-Form:

```
const registerValidation = (data) => {
 const schema = Joi.object({
   firstName: Joi.string().required(),
   lastName: Joi.string().required(),
   gender: Joi.string().required(),
   birthDate: Joi.string().required(),
```

```
    email: Joi.string().required().email(),
    password: Joi.string().required().min(6),
  });

  return schema.validate(data);
};
```

Quellcode 6: Vereinfachtes Schema zur Validierung der Eingabe

Man kann sehen, dass alle Eingaben verpflichtend sind und ein String erwartet wird. Zusätzlich wird noch überprüft, ob die E-Mail korrekt eingegeben wurde und ob das Passwort mehr als 6 Stellen hat.

7.3.3. Sicherheit

Wie bereits erwähnt, muss der Nutzer authentifiziert sein, damit er auf gewissen Routen zugreifen kann. Sobald sich der Nutzer erfolgreich eingeloggt hat, wird ein JSON Web Token (JWT) generiert. Dieser wird bei jeder Anfrage mitgesendet, sodass unser Backend weiß, dass der Nutzer authentifiziert ist. Bei jeder Anfrage an einen Endpunkt wird zuerst überprüft, ob der korrekte JWT mitgeliefert wurde, bevor die Anfrage bearbeitet wird.

Dies funktioniert mithilfe des sogenannten Tokensecrets. Das ist eine beliebig gewählte Zeichenfolge, welche zur Erstellung des JWTs verwendet wird. Bei einer Anfrage an das Backend wird dann überprüft, ob die Tokensecrets übereinstimmen. Ist dies der Fall, kann die Anfrage gestellt werden, ansonsten wird ein Error ausgegeben.

1 Bei Anmeldung wird ein JSON Web Token im File Auth.js erstellt.

```
TOKEN_SECRET = aEFN4J45FMWEE!-.SDF(WEGT3
const token = jwt.sign({ _id: user._id }, process.env.TOKEN_SECRET, { expiresIn: '1h' });
```

2 Vor jeder Abfrage wird überprüft ob Nutzer berechtigt ist.

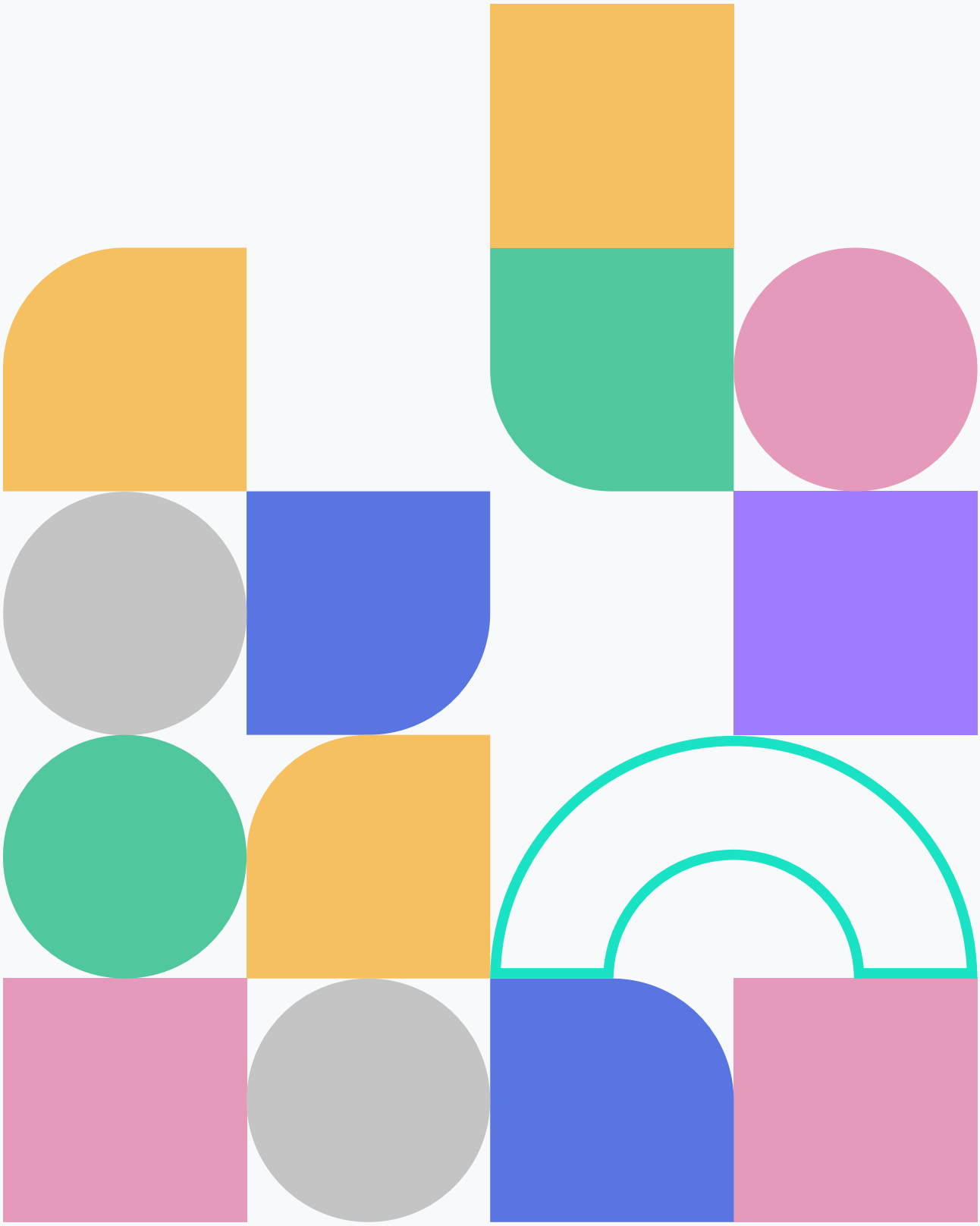
```
router.get('/', verify, async (req, res) => {
   // Abfrage an Datenbank
});
```

3 JSON Web Token wird verifiziert.

```
const verified = jwt.verify(token, process.env.TOKEN_SECRET);
```

Quellcode 7: Erstellung & Verifizierung eines JWT

8. FRONTEND



8.1. RÜCKBLICK

In der Dokumentation des Konzeptionssemesters haben wir uns für das JavaScript-Framework React zur Umsetzung des Frontends entschieden. Diese Entscheidung haben wir jedoch direkt zu Beginn des Produktionssemesters gekippt. Wir haben uns entschlossen, Vue.js anstatt React zu verwenden. Einer der Hauptgründe für die Änderung war, dass bei Vue.js HTML, CSS und JavaScript als Sprachen verwendet werden.

Bei React muss man mit JSX einen neuen Syntax erlernen. Dies hat vor allem unseren nicht Informatik-affinen Gruppenmitgliedern den Einstieg erleichtert.

Die Art der App ist mit einer Progressive Web App die gleiche geblieben, wie in der Dokumentation des Konzeptionssemesters vorgeschlagen.

8.2. VUE.JS

Abb. 66: Logo Vue

Vue ist ein progressives JavaScript-Framework. Progressiv bedeutet hierbei, dass Vue auch nur in bestimmte Teile einer Applikation eingebaut werden kann. Vue unterteilt die Nutzeroberfläche in einzelne Komponenten und nutzt ein Virtual Document Object Model (VDOM). Das ist eine Kopie des tatsächlichen Document Object Models (DOM). Das DOM ist die Spezifikation der Schnittstelle zwischen HTML und JavaScript. Die Darstellung aller HTML-Dokumente erfolgt in einer Baumstruktur. Die Knoten in diesem Baum stehen für einzelne HTML-Objekte, wie z. B. eine Überschrift. Das VDOM ist auf ein Minimum an Daten beschränkt, dadurch ist die Durchführung von Änderungen schnell möglich. Zur Implementierung der Komponenten wird HTML, CSS und JavaScript genutzt. Der Vorteil dabei ist, dass sich Webentwickler nicht erst in den JSX-Syntax wie z. B. bei React einarbeiten müssen. Sie können direkt mit dem Erstellen einer Applikation beginnen.[12, 13]

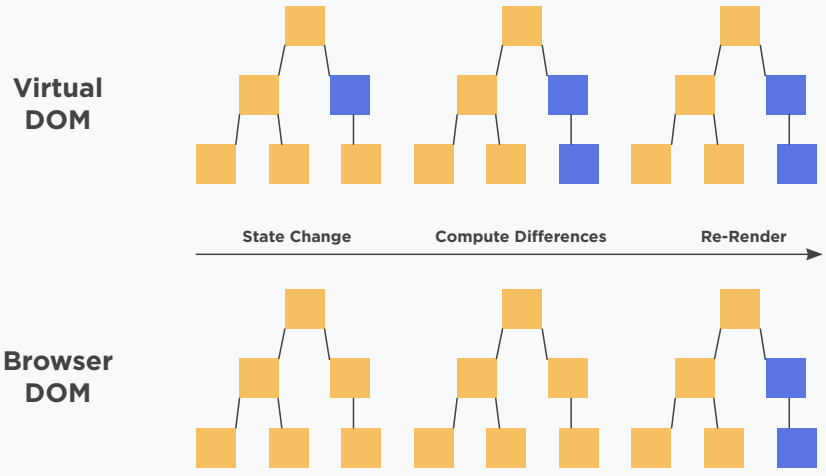


Abb. 67: Vergleich DOM vs VDOM

Vue basiert auf dem sogenannten Model-View-View-Model (MVVM) und verwendet eine bi-direktionale Kommunikation. Dabei verwaltet das Model die Daten, während die View für die Darstellung zuständig ist. Die dazwischenliegende Vue-Instanz ist das View-Model. Sie bildet die Brücke zwischen Model und View. Werden Daten in der View geändert, sorgt es für eine Synchronisierung mit dem Model.[14, 15]

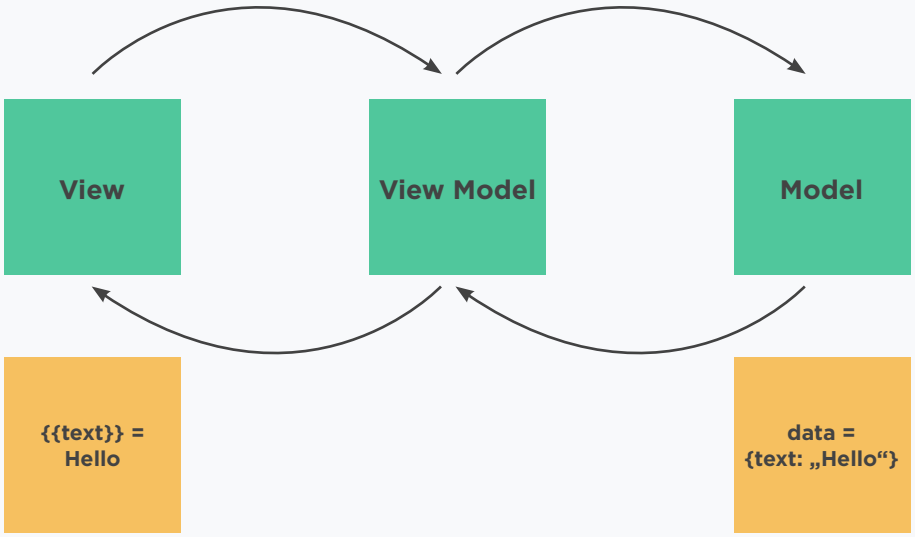


Abb. 68: Model-View-ViewModel

8.2.1. Vuex und Local Storage

Vuex ist eine State Management Bibliothek für Vue.js. Der Vuex-Store dient als zentraler Speicherplatz für alle Komponenten der Anwendung. Vorgefertigte Pattern sorgen dafür, dass die Zustände nur in einer vorhersehbaren Weise mutiert werden können. Der Einsatz von Vuex ist vor allem hilfreich, wenn mehrere Komponenten einen gemeinsamen Zustand teilen. So muss der Zustand nicht über mehrere Komponenten im Komponentenbaum weitergegeben werden, sondern jede Komponente kann den Zustand im zentralen Store abfragen.[16]

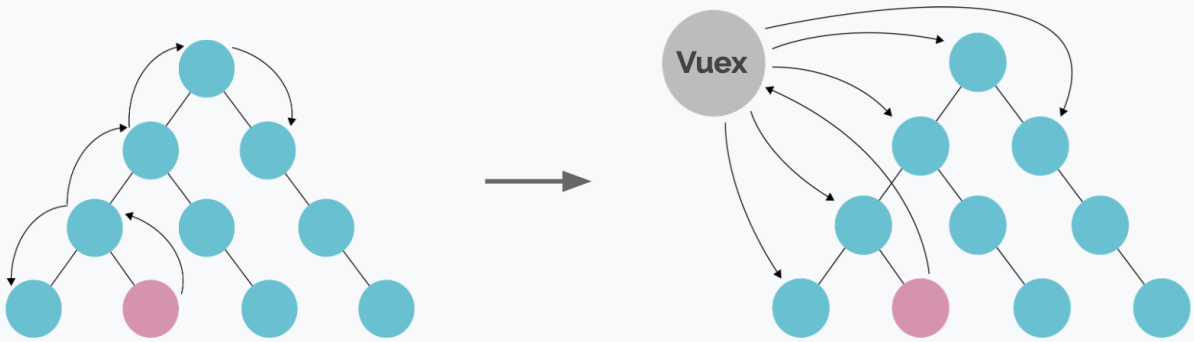
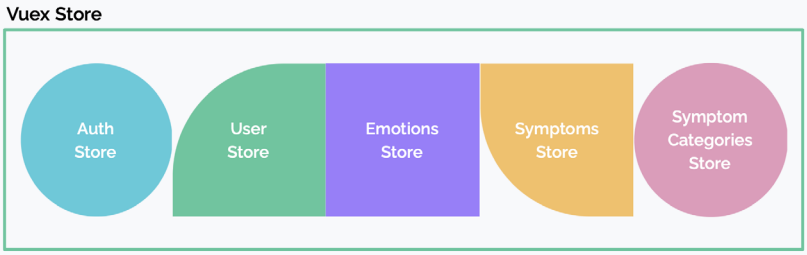


Abb. 69: Normales State Management vs. State Management mit Vuex

In unserem Fall speichern wir fast alle Daten aus unserer Datenbank im Vuex-Store. Damit der Store übersichtlich bleibt, haben wir ihn in Module unterteilt. Wir haben für jeden API Endpunkt ein eigenes Modul erstellt. Daraus ergibt sich folgende Aufteilung für unseren Store:



Jedes Modul hat einen State und Mutations, Actions und Getter. Der Aufbau ist dabei immer gleich. Jeder API Endpunkt hat eine eigene Action, welche die Daten aus der Datenbank holt. Innerhalb dieser Action werden die Mutations aufgerufen. Diese ändern den State. Mithilfe von Gettern können die Komponenten auf den State des Moduls zugreifen.

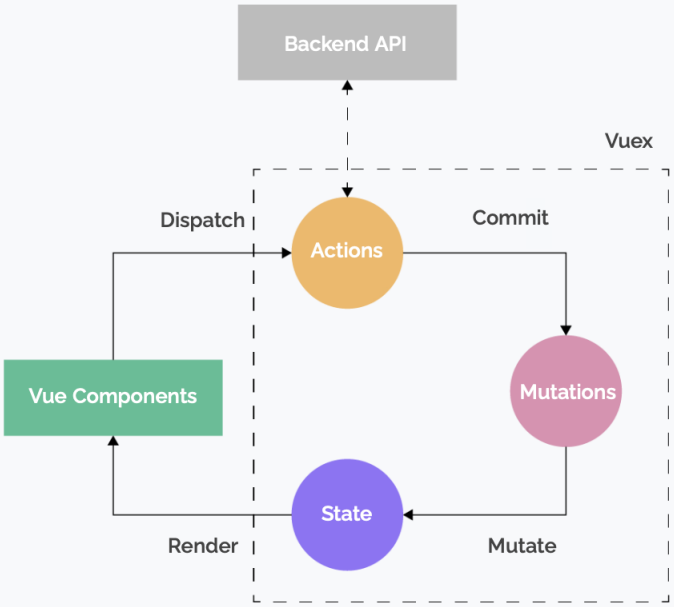


Abb. 70: Vuex Statemanagement Ablauf

Anhand eines Teils des Emotion-Moduls soll im Folgenden der grundlegende Aufbau eines Moduls erklärt werden.

```
// action types
export const GET_ALL_EMOTIONS = „getAllEmotions“;

// mutation types
export const GET_ALL_EMOTIONS_START = „getAllEmotionsStart“;
export const GET_ALL_EMOTIONS_SUCCESS = „getAllEmotionsSuccess“;
```

```
export const GET_ALL_EMOTIONS_ERROR = „getAllEmotionsError“;

export default ({
 state: {
   status: “,
   emotions: [],
 },
 getters: {
   getUserEmotions(state) {
     return state.emotions;
   },
 },
 actions: {
   [GET_ALL_EMOTIONS]: async ({ commit, rootState }) => {
     const userid = rootState.user.id;
     commit(GET_ALL_EMOTIONS_START);
     await axios.get(„/api/emotions/“, { params: { userid } })
       .then((resp) => {
         commit(GET_ALL_EMOTIONS_SUCCESS, resp);
       })
       .catch(() => {
         commit(GET_ALL_EMOTIONS_ERROR);
       });
   },
},
 mutations: {
   [GET_ALL_EMOTIONS_START]: (state) => {
     state.status = „loading“;
   },
   [GET_ALL_EMOTIONS_SUCCESS]: (state, resp) => {
     state.status = „success“;
     state.emotions = resp.data;
   },
   [GET_ALL_EMOTIONS_ERROR]: (state) => {
     state.status = „error“;
   },
 },
 modules: {},
});
```

Quellcode 8: Vereinfachter Aufbau eines Vuex-Moduls

Zuerst haben wir alle Action- und Mutationstypes als Konstanten definiert. So konnten wir diese exportieren und überall wiederverwenden.

Im State halten wir die Daten aus der Datenbank, in diesem Fall die Einträge des Nutzers im Modul Emotionen. Außerdem wird der Status der Mutation festgehalten. Dieser zeigt an, ob die Mutation des States erfolgreich war oder nicht.

Die Getter werden genutzt, um den State zurückzugeben. In diesem Fall wird der aktuelle State der Emotionen zurückgegeben.

Die Action wird immer aus einer Komponente heraus aufgerufen. Anschließend wird mithilfe des HTTP-Clients Axios eine Anfrage an unsere API gesendet, alle Emotionen des Nutzer zurückzugeben. Dabei wird auf den State des User-Modules zugegriffen, um die ID der Nutzers zu bekommen. So wird sichergestellt, dass nur die Einträge des angemel-

deten Nutzers zurückgegeben werden. Wenn die Anfrage an die Datenbank erfolgreich war und Daten zurückgegeben wurden, wird die entsprechende Mutation ausgeführt. Diese aktualisiert die Daten und setzt den Status auf “Success”. Ist die Abfrage nicht erfolgreich gewesen wird der Status auf “Error” gesetzt.

Normalerweise wird der State nach jedem Neuladen der Applikation gelöscht. Damit wir die Daten nicht jedes Mal erneut aus der Datenbank abfragen müssen, haben wir das Plugin Persisted State verwendet. Die Daten werden so nur noch gelöscht, wenn sich der Nutzer ausloggt oder den Tab mit unserer Applikation schließt.

In allen Actions verwenden wir für die Abfrage nur relative URLs, im Beispiel an den Endpunkt “/api/emotions”. Axios ermöglicht es uns, eine sogenannte BaseURL festzulegen. Sie wird im File main.js der Vue-Applikation festgelegt. Dies hat den Vorteil, dass wir die URL schnell ändern können, z. B. wenn wir unsere Applikation auf den Server laden.

```
axios.defaults.baseURL = „http://fine-app.multimedia.hs-augsburg.de/“;
```

Quellcode 9: Festlegen einer BaseURL mit Axios

Jeder Browser besitzt einen Local Storage. Das ist ein lokaler Speicherplatz, in dem die Daten von Webanwendungen über mehrere Sessions hinweg gespeichert werden können. Die Daten bleiben so lange im Local Storage, bis sie von der Webanwendung wieder entfernt werden. Wichtig zu erwähnen ist, dass im Local Storage nur Strings gespeichert werden können. Will man z. B. ein Objekt speichern, muss man dieses zuvor in einen String konvertieren.[17]

Wir geben den JWT, welcher nach der Anmeldung im Backend generiert wird, in den Local Storage. Direkt nach der Anmeldung wird dieser Token auch im Axios-Header gesetzt. So wird der Token bei jeder Anfrage mitgesendet und der Nutzer kann sich im Backend authentifizieren.

```
// Vuex Store, auth.js - store token in local storage and update axios
header localStorage.setItem(„user-token“, authtoken);
axios.defaults.headers.common.Authorization = authtoken;
```

Quellcode 10: Token im Axios Header setzen und im Local Store ablegen

Ein Neuladen der Anwendung setzt den Axios-Header zurück. Deshalb wird im File main.js der Vue-Applikation immer überprüft, ob ein Token im Local Storage vorhanden ist. Wenn dies der Fall ist, wird er als Axios-Header gesetzt. So ist sichergestellt, dass sich der Nutzer auch nach einem Neuladen der Seite nicht erneut einloggen muss.

```
// main.js - Set Authorization header when page is reloaded and user log-
ged in const token = localStorage.getItem(„user-token“);
if (token) {
 axios.defaults.headers.common.Authorization = token;
}
```

Quellcode 11: Token im Axios Header setzen nach Neuladen der SeiteLocal Store ablegen

8.2.2. Vue Router

In unserer App nutzen wir den Vue Router. Dies ist die offizielle Bibliothek für Routing, welche von dem gleichen Team wie Vue.js erstellt und gewartet wird. Der Router wird bei einer Single Page Application dafür benötigt, dass die View der Applikation mit der Adresszeile synchronisiert wird.

Bei unserer Applikation sind /login und /register die einzigen Routen, bei denen der Nutzer nicht authentifiziert sein muss, um darauf zuzugreifen. In der File index.js des Routers wird bei jedem Wechsel der Route überprüft, ob der Nutzer authentifiziert ist. Dies geschieht mit einer Abfrage an das Auth-Module in unserem Store. Um die Ladezeit der Applikation gering zu halten, werden die benötigte Komponenten erst geladen, wenn die entsprechende Route aufgerufen wird.

Hier ist eine Übersicht all unserer Routen:

Nutzer ist nicht authentifiziert:
- /register - Nutzer kann sich registrieren
- /login - Nutzer kann sich anmelden

Nutzer ist authentifiziert:
- /dashboard - zeigt das Dashboard an
- /calendar - zeigt den Kalender an
- /statistics - zeigt die Statistik an
- /settings - zeigt die Einstellungen an
- /module-entry/:entryModule/:id - Nutzer kann einen Eintrag bearbeiten

8.2.3. Stylesheets

Für unser Styling verwenden wir Sass. Sass erweitert CSS und hat daher viele zusätzliche Features, wie die Möglichkeit, Variablen zu verwenden oder Funktionen zu schreiben.
Jede Vue-Komponente hat ihr eigenes SCSS-File, das jeweils an das Vue-File gescoped sind und somit auch nur greift, wenn dieses Vue-File angezeigt wird. Grund-Stylings, wie Farben, Schriften, Buttons oder das Grundlayout haben wir in eigene Dateien gepackt und referenzieren diese in jedem Style-Sheet.[18]

Da wir je nach Modul verschiedene Farben haben und diese auch im Frontend brauchen, gibt es für jedes Modul eigene Klassen, die dann nach Bedarf die background color oder color setzen. Genauso gibt es für die einzelnen Text-Stile, die im Design verwendet werden eigene Klassen. So können die Variablen flexibel in Vue verwendet werden.

8.3. AUFBAU DER KOMPONENTEN

Damit alles übersichtlich bleibt, haben wir unsere Views, also die Start-Komponenten jedes Routings, in einen extra Ordner gepackt. Alle anderen Komponenten sind im Ordner components und je nach Bedarf ebenfalls noch in Unterordner verpackt. Vorweg muss gesagt werden, dass wir alle noch keine Erfahrungen mit Vue 3 gemacht haben. Wir haben dennoch versucht, unsere Komponenten möglichst modular und wiederverwendbar aufzubauen. Beim Entwickeln haben wir viel gelernt und man kann vermutlich direkt jetzt die ersten Zeilen nochmal umschreiben.

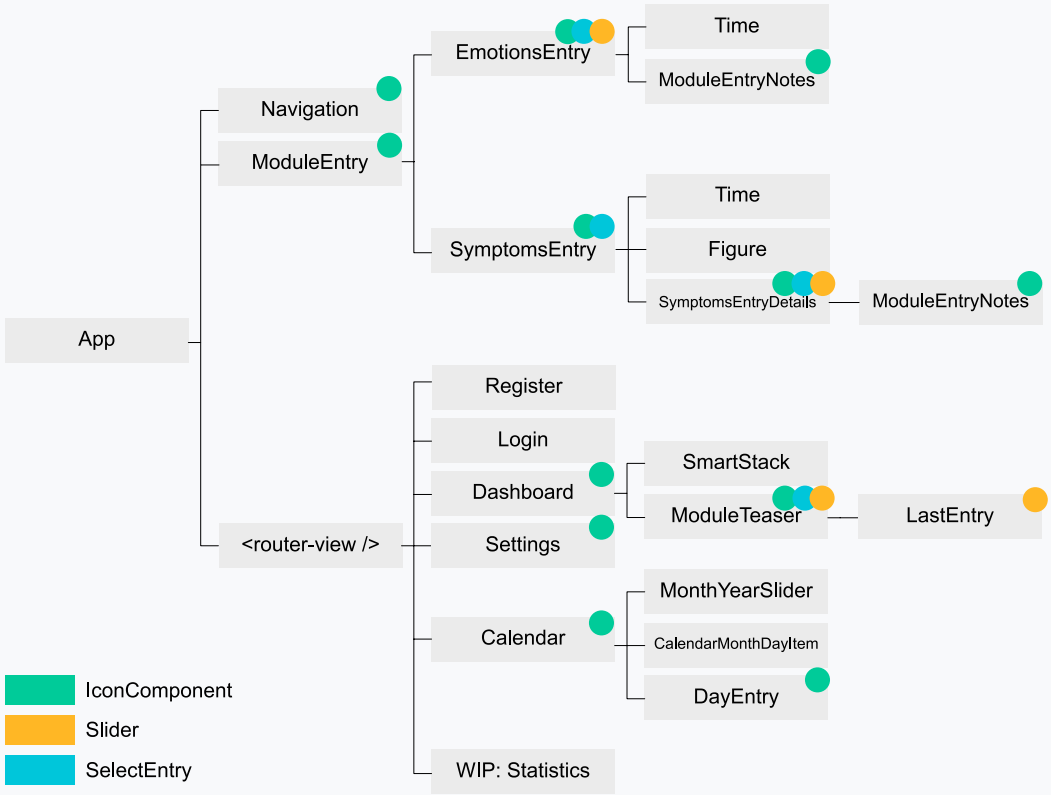


Abb. 71: Aufbau der Komponenten

8.3.1. Kalender (Calendar.vue)

Die Grundlage für unseren Kalender bildet das Tutorial Let's Make a Vue-Powered Monthly Calendar von css tricks (https://css-tricks.com/lets-make-a-vue-powered-monthly-calendar/). Da wir in unserem Design aber eine Wochenausgabe brauchen, haben wir das Tutorial nochmal komplett umgeschrieben und so an unser Design und unsere Bedürfnisse angepasst.[19]

Klickt man auf einen bestimmten Tag und hat dieser Tag Einträge der Module, öffnet sich eine Übersicht aller Einträge dieses Tages. Die Einträge werden alle am Anfang beim Laden des Kalenders geladen und bei Klick auf den einzelnen Tag zugeordnet. Die Übersicht der Ein-

träge werden nach der jeweiligen Woche angezeigt. Dazu greifen wir in das Monats-Array mit den einzelnen Tagen ein und setzen nach jeder Woche ein weiteres Element, dass wir dann im CSS stylen können und mit den jeweiligen Einträgen füllen können.

Den Datums- und Jahres-Switcher oberhalb des Kalenders haben wir ebenfalls auf unser Design angepasst. Für eine Vereinfachung der Datumsberechnung und -ausgabe verwenden wir die Bibliothek dayjs (https://day.js.org/).[20]

8.3.2. Dashboard (Dashboard.vue)

Das Dashboard besteht aus dem Smart-Stapel und einzelnen Modul-Teasern. Diese werden, je nachdem welche Module in den Einstellungen ausgewählt werden, im Dashboard angezeigt. Hier kann die Intensität als Schnelleingabe eingestellt werden. Außerdem kann hier schnell ein neuer Eintrag gemacht werden. Dieser ist allerdings noch nicht implementiert. Auch das Wischen, um direkt zum Kalender oder zur Detaileingabe zu kommen, welches im Design vorgesehen ist, ist noch nicht umgesetzt.
Es wird in jedem Modul der letzte Eintrag aus der Datenbank geholt und angezeigt. Dieser kann auch bearbeitet werden.

8.3.3. Eingabe (ModuleEntry.vue)vue)

Ohne Eingabe funktioniert unsere App nicht und deshalb war dies eine der ersten Komponenten, die wir umgesetzt haben. Wir haben uns dabei für unseren Prototypen auf die Symptom- und die Gefühl-Eingabe fokussiert.
Für die Bearbeitung der Einträge wird die gleiche Komponente verwendet. Hier wird dann lediglich eine ID mit übergeben und anschließend wird dieser Eintrag aus der Datenbank geholt und alle Felder werden vorausgefüllt.

In jeder Eingabe kann das Datum und die Zeit angepasst werden. Das wird in der Time-Komponente (Time.vue) mit Hilfe des HTML-Input-Elements mit dem Typ date bzw. time gesetzt. Die jeweiligen Daten aus der Datenbank werden in den Unterkomponenten der jeweiligen Module geholt.

8.3.4. Symptome (SymptomsEntry.vue)

Die Detail-Eingabe eines Symptoms erfolgt durch Auswählen einer Kategorie und anschließendem Setzen des Ortes auf einer Abbildung eines Menschen. Diese Abbildung ist je nach angegebenem Geschlecht weiblich oder männlich und ist als SVG eingebunden. Um den genauen Ort auch als String verwenden zu können, haben wir in das SVG der Abbildung einzelne Zonen mit einer bestimmten id hinterlegt. So kann bei Klick auf die Schulter genau bestimmt werden, ob es sich um die linke Schulter oder die rechte Schulter

handelt. Außerdem können genaue Zonen am Kopf, was bei der Migräne wichtig ist, genau beschrieben werden. Diese Figur wird als inline-SVG gespeichert, um hier die volle Kontrolle zu behalten, damit aber die Dateien übersichtlich bleiben, haben wir sie in eine eigene Komponente (Figure.vue) geschrieben. Je nach Ansicht (vorne oder hinten) und Geschlecht (weiblich oder männlich) wird das jeweilige SVG ausgegeben.
Sobald der Nutzer auf die Abbildung klickt, wird an dieser Stelle ein Punkt mit Koordinaten gesetzt, die auch in der Datenbank gespeichert werden. Außerdem werden noch der vorhin beschriebene String der genauen Location gespeichert sowie ob es sich um die Vorder- oder Rückansicht handelt. Zunächst pulsiert der Punkt, bis man die Intensität setzt. Dafür öffnet sich am rechten Rand ein Overlay, auf dem die Intensität von eins bis fünf gesetzt werden kann. Anschließend wird dieser Eintrag in der Datenbank gespeichert. Klickt man nun auf diesen Punkt, öffnet sich ein weiteres Overlay (SymptomsEntryDetails.vue), in dem die Details, wie Notizen oder weitere Schlagworte eingetragen werden können. Da die Eingabe der Details auch in allen anderen Modulen vorkommt, haben wir diese in eine eigene Komponente geschrieben (ModuleEntryNotes.vue). Damit auch die Details für den jeweiligen Eintrag richtig geladen werden, wird direkt nach Setzen des Punktes die id, die in die Datenbank eingetragen wurde, zum gesetzten Punkt hinzugefügt. Klickt der Nutzer nun auf diesen Punkt, kann im Frontend die id ausgelesen werden und die Details aus der Datenbank werden korrekt ausgegeben oder können dem richtigen Eintrag zugeordnet werden.

Klickt der Nutzer auf Abbrechen, werden die aktuellen Einträge wieder aus der Datenbank entfernt. Da die Punkte dynamisch über Javascript gesetzt werden und somit kein Vue-Element sind, werden sie direkt in die Datenbank geschrieben und müssen somit bei Abbruch wieder gelöscht werden. Alle Einträge, die während der Eingabe gemacht werden, werden in ein Array gespeichert, dass dann anschließend auch geleert wird.

Mit Hilfe von Events können die Komponenten untereinander kommunizieren. Damit bei einem neuen Eintrag die Seite nicht neu geladen werden muss, um die neuen Einträge auch im Kalender anzuzeigen, haben wir einen globalen Emitter mit Hilfe der externen Bibliothek mitt (https://github.com/developit/mitt) eingebaut. Diese wird auch in der Dokumentation von Vue 3 empfohlen (https://v3.vuejs.org/guide/migration/events-api.html#root-component-events). [21, 22]

8.3.5. Gefühle (EmotionsEntry.vue)

Die Eingabe der Gefühle ist wesentlich einfacher aufgebaut als die Eingabe der Symptome. Hier sind lediglich ein Slider, der die Intensität festlegt, sowie die Komponente SelectEntry für die Eingabe von Gefühlen und weiteren Vorkommnissen, eingebaut. Außerdem kann der Nutzer Notizen, Fotos und Sprachnotizen eintragen. Fotos und Sprachnotizen sind allerdings noch nicht programmiert.

Nachdem der Nutzer den Eintrag speichert, wird dieser, anders wie bei Symptome, in die Datenbank geschrieben.

8.3.6. Icons (IconComponent.vue)

Um die Verwendung von SVG-Icons so einfach wie möglich zu machen, haben wir eine eigene Komponente dafür eingebaut. Alle unsere Icons befinden sich im Ordner src/assets/icons. Sie sind wie folgt benannt und haben alle die Farbe #434343.

[Größe]-[Name].svg

Mit Hilfe des npm packages svg-sprite (https://www.npmjs.com/package//svg-sprite) werden alle Icons in ein Sprite gepackt. So müssen diese nicht einzeln geladen werden. In der package.json geben wir den Pfad zu den Icons an. Außerdem geben wir die Farbe an, die ausgetauscht werden soll. Damit man die Farbe der SVGs im css stylen kann, muss diese mit currentColor hinterlegt sein. Mit dem Befehl npm run icons wird zunächst das Sprite gebaut und anschließend die angegebene Farbe, in unserem Fall #434343, mit currentColor ersetzt.[23]

Hinweis: Da unsere Einstellungen in der package.json nur auf Mac-Geräten funktioniert hat, haben wir hier für Windows-Geräte einen eigenen Befehl erstellen müssen: npm run icons-win.

Der IconComponent können folgende Einstellungen übergeben werden:
- size, also die Höhe, die auch im Icon-Namen selbst hinterlegt ist
- width, falls die Breite abweicht
- name
- color, entweder als #-Farbe oder die Klasse der Farbe (z. B. fine-grey-dark)
- strict mode, gibt an, ob das Icon in seiner Größe verändert werden kann

Und das kann dann wie folgt aussehen:

```
<IconComponent name="trash" :size=24 color="fine-signal" />
```

Quellcode 12: Icon Komponente

Das Icon selbst wird dann innerhalb eines span-Elements ausgegeben und ist so flexibel an jeder Stelle einsetzbar.

8.3.7. Select Entry (SelectEntry.vue)

Die Select-Entry-Komponente kommt im Design an vielen Stellen vor und deshalb haben wir sie in eine Komponente gepackt. Je nachdem ob multiselect erlaubt ist oder nicht, ist die Funktionalität etwas anders, deshalb ist diese Komponente sehr komplex aufgebaut. Eventuell macht es an dieser Stelle sogar Sinn, die Komponente in zwei aufzuteilen, da sich die Funktionalität bei Multiselect stark unterscheidet. Das wäre eine Überlegung, die man zur Optimierung der Struktur vornehmen könnte.

Übergeben werden können folgende Einstellungen:

- module (das jeweilige Modul zur korrekten Ausgabe der Farben)
- buttonLabel (was bei geschlossenem Zustand dran stehen soll)
- multiselect (true oder false, je nachdem ob Mehrauswahl möglich sein soll oder nicht)
- list (eine vordefinierte Liste, die bereits Einträge enthält)

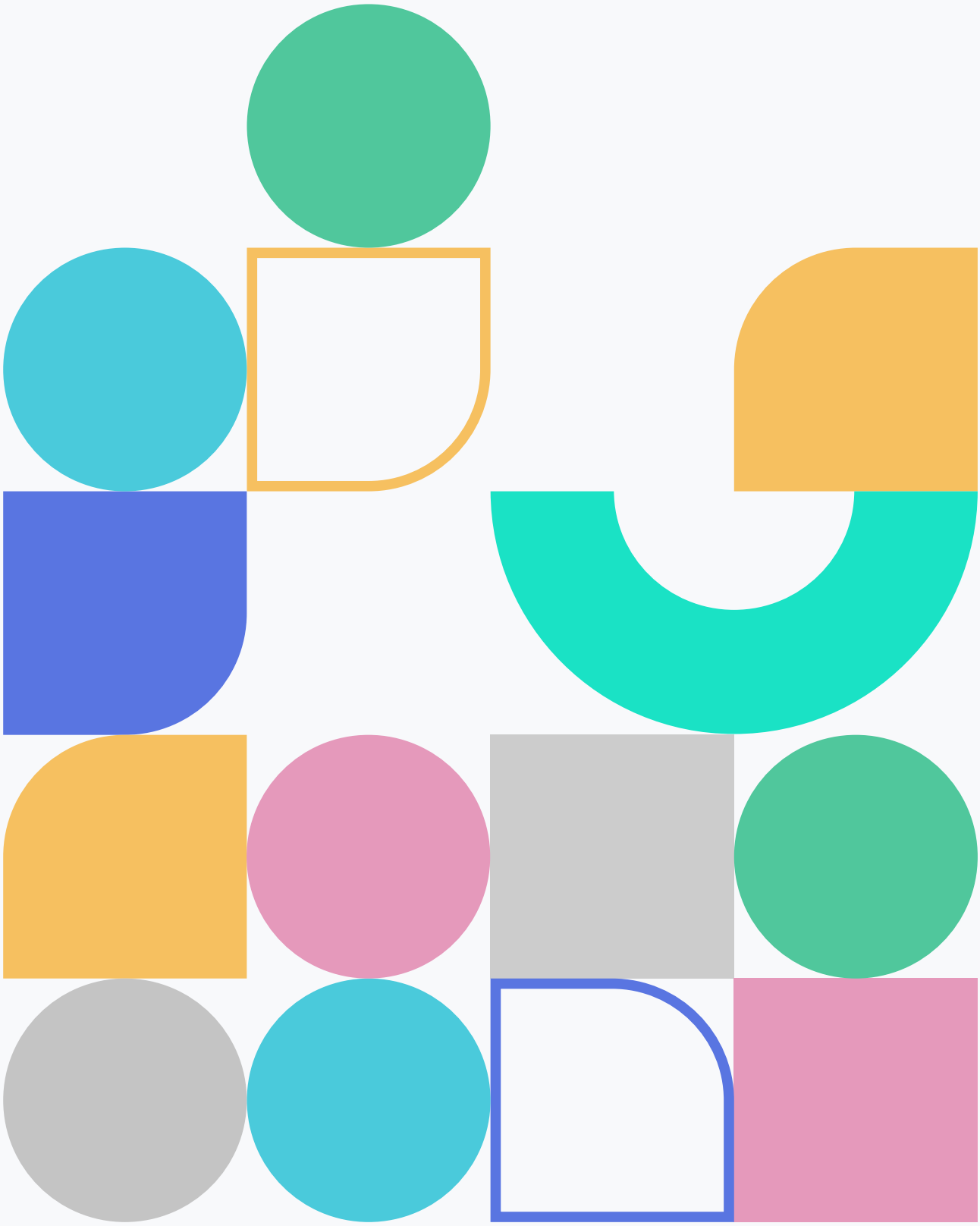
Die vordefinierte Liste beinhaltet neben dem Namen der Option auch, ob die jeweilige Option bereits ausgewählt ist. So können bereits gesetze Einträge (zum Beispiel beim Bearbeiten eines Eintrags) angezeigt werden.

8.3.8. Einstellungen (Settings.vue)

In den Einstellungen hat der Nutzer die Möglichkeit, seine Profildaten anzupassen. Lediglich das Ändern des Passwortes muss hier noch angepasst werden, da aus der Datenbank der gehashte Wert gesetzt wird.

Außerdem hat der Nutzer hier die Möglichkeit Module an- und abzuwählen. Welche Module ausgewählt sind, werden in der Datenbank gespeichert.

9. SPRACHASSISTENT



9.1. KONZEPT

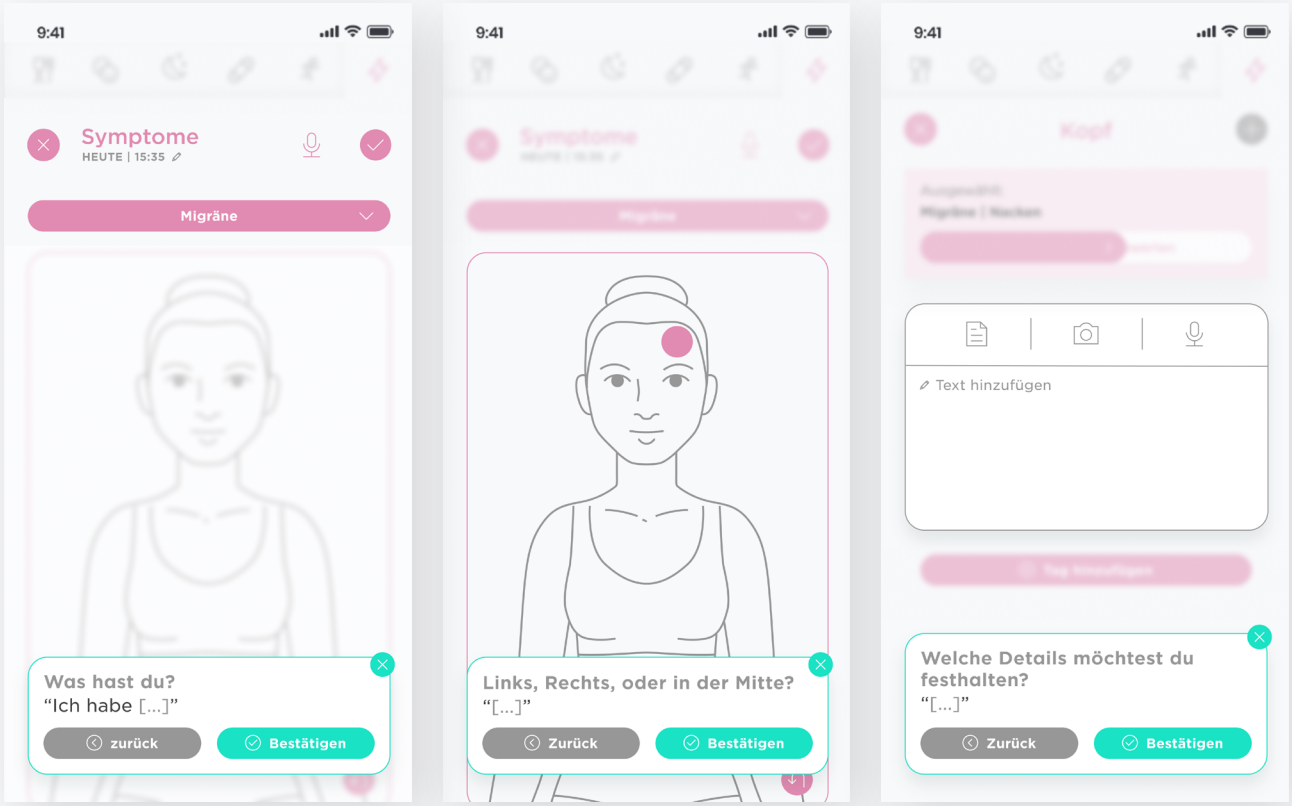


Abb. 72-74: Spracheingabe

In der Konzeption war schon lange die Rede von einer Möglichkeit, noch viel schneller Eingaben zu tätigen, ohne sich durchklicken zu müssen. Dafür angedacht ist ein Sprachassistent, welcher den Nutzer unterstützt und eine Eingabe ohne ewiges Tippen ermöglicht. Geplant ist, dass dieselben Schritte wie beim bisherigen Eingeben durchlaufen werden.

Um den Sprachassistenten zu starten, klickt der Nutzer auf das Mikrofon-Icon. Um eine entsprechende Rückmeldung zu erhalten, dass der Assistent gestartet wurde, soll ein Kreis um das Mikrofon pulsieren, wie es schon die Punkte auf der Figur im Prototypen im Modul Symptome tun. Bei jedem Schritt werden unwichtige visuelle Komponenten unscharf, sodass klarer ist, welche Änderungen getätigt werden. Nun gibt es verschiedene Use Cases des Sprachassistenten. Neulinge werden jeden einzelnen Screen durchlaufen müssen wie oben in der Abbildung, wo ganz genau durch die Eingabe durchgeführt wird und was zum entsprechenden Teil gesagt werden muss. Im Idealfall soll der Nutzer bereits wissen, was er alles erwähnen muss, das wichtig ist. So könnte er einen Satz sagen wie: “Ich habe Schmerzen am linken Handgelenk mit einer Intensität von vier”. Hier müsste in der Umsetzung darauf geachtet werden, dass alle Worte erkannt und richtig Kategorien zugeordnet werden.

9.2. UMSETZUNG

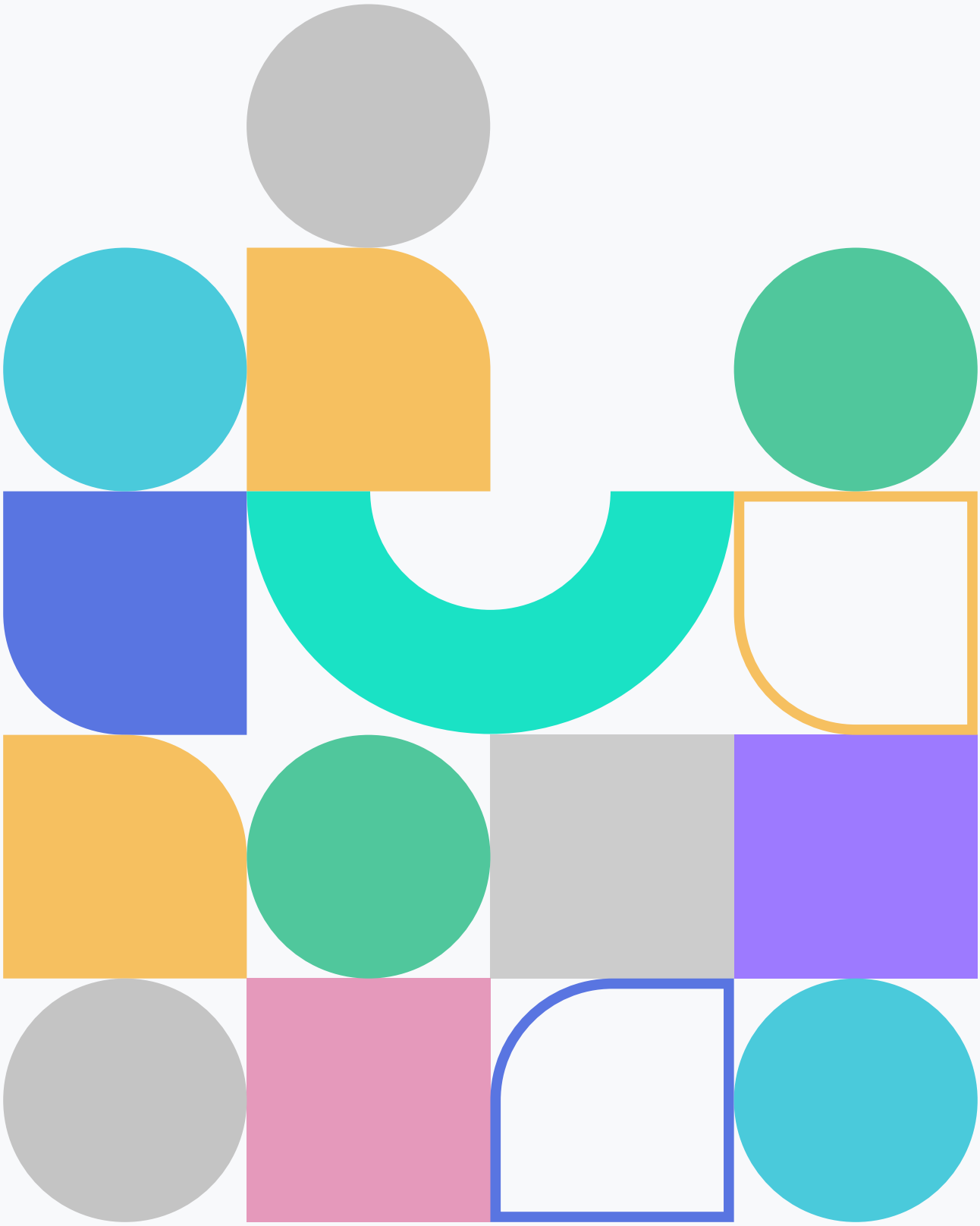
Die Spracheingabe könnte z. B. mithilfe der Web Speech API umgesetzt werden. Sie ist eine Spezifikation des World Wide Web Consortiums und stellt durch eine im Browser integrierte JavaScript-Programmierschnittstelle unter anderem die Funktionalität zur Spracherkennung bereit. Die API wird aktuell leider nur von Google Chrome und Microsoft Edge standardmäßig unterstützt.[24, 25, 26, 27]

Weitere Möglichkeiten zur Umsetzung bieten die Cloud Speech-to-Text API von Google oder die JavaScript Bibliothek Pocketsphinx.js.[28, 29]

Wir haben bereits einige Tests mit der WebSpeech API durchgeführt. Das Problem ist jedoch, dass der Nutzer in unserem Fall durch den Prozess geleitet wird und bestimmte Keywords erkannt werden müssen. Dafür ist die Webspeech API nicht ausgelegt. Eigentlich bräuchte man hierzu eine KI, welche auch bei leichten Abweichungen vom Standardtext noch zuverlässig die Keywords erkennt.

Durch unsere Tests konnten wir jedoch feststellen, dass eine Spracheingabe durchaus eine gute Alternative zur händischen Eingabe sein kann. Besonders bei häufiger Verwendung der App, kann der Nutzer damit sehr schnell seine Eingaben erledigen.

10. TESTS



Zum Abschluss standen Nutzertests an. Um diese richtig durchzuführen, würden wir gerne Langzeittests durchführen. Mit einfachen Interviews haben wir uns einen generellen Überblick über den Status quo des aktuellen Prototypen geschaffen und planen Tests zu einem späteren Zeitpunkt, wenn wir eventuell an dem Projekt weiterarbeiten. Für die Interviews mit Nutzern haben wir fünf Probanden um ihre Meinung gefragt und bereits eine Menge Einblicke erhalten. Die Probanden sind entsprechend unserer Zielgruppe ausgewählt.

Auch für diese Interviews wurde ein Leitfaden erstellt. Getestet wurde via Zoom. Im Browser wurde die Anwendung in Smartphone Größe gestartet. Die Vorgehensweise war immer, zunächst den Click-Dummy des Tutorials als Einführung in Figma zu zeigen. Hierbei geht es nicht um den Vorgang der Registrierung oder des Logins, sondern einen Überblick über die Features zu verschaffen. Diese sind in fine nun mal nicht immer selbsterklärend. Hier fanden wir sehr viel Bestätigung in den Interviews, dass das Tutorial benötigt wird und auch von der Länge und den Inhalten pro Erklärung völlig im Rahmen war. Wer dennoch keine Lust hat, kann diese Erklärung, die nur beim ersten Öffnen der App erscheint, auch überspringen.

Der Rest des Interviews bestand aus dem Ausprobieren und Testen des Prototypen anhand von diesen drei Szenarien:

1. Gehe in die Einstellungen und füge die Module Symptome und Gefühle hinzu
2. Füge Symptome/Gefühle hinzu, übe Spracheingabe reden
3. Wähle den heutigen Eintrag im Kalender aus

Das erste Szenario zeigt, dass die Probanden keinerlei Probleme hatten, die Einstellungen zu finden. Auch das Einschalten von Modulen gestaltet sich unproblematisch. Was gleich als erstes angesprochen wurde, war aber der Datenschutz. Was passiert mit den Daten, wo werden diese gespeichert? Hier müssen wir ganz klar sagen, dass die Daten auf den Servern gespeichert werden müssen, allerdings wollen wir laut dem ausgearbeiteten Konzept des Ethikkurses darauf achten, dass höchste Sicherheitsstandards gelten, sollte die App wirklich veröffentlicht werden.

Weiter fällt auf, dass die größten Probleme tatsächlich beim Hinzufügen von Einträgen liegen. Häufig ist unklar, wann eine Kategorie oder ein Schlagwort tatsächlich angenommen wurden und erst nach mehrmaligen Versuchen war die Funktionsweise klar. Vorgeschlagen wurde ein animierter Pfeil der darauf aufmerksam macht, dass hier das Pluszeichen geklickt werden muss. Auch der Bestätigungsbutton ist von der Position nicht optimal, da viele am Ende des Screens nach einer Möglichkeit zum Speichern suchen. Dadurch wurde der Button zum Drehen der Figur als ein Zurück-Button oder zum Löschen des Eintrags verstanden. Kleine Umpositionierungen könnten hier vielleicht schon für Klarheit sorgen. Das Icon zum Verfassen von Texten neben der Kamera für das Hochladen von Fotos und der Aufnahme von Sprachnotizen wirkte bei einer Person wie ein Button zum Hochladen von Dokumenten. Auch hier kann weiter an der Iconsprache gearbeitet werden. Das Mikrofon wurde richtig als Spracheingabe erkannt und jedoch würden nicht alle Probanden die Methode selbst nutzen, weil sie grundsätzlich von solcher Technik absehen. Aber die Nützlichkeit für viele andere Menschen wurde erkannt und die Möglichkeit als sinnvoll gehalten. Als Vorschlag wurde genannt, entsprechende Fragen vorzulesen, bevor die Person diese beantworten kann, sodass nicht selbstständig lesen muss oder sich Fragen erschließen muss.

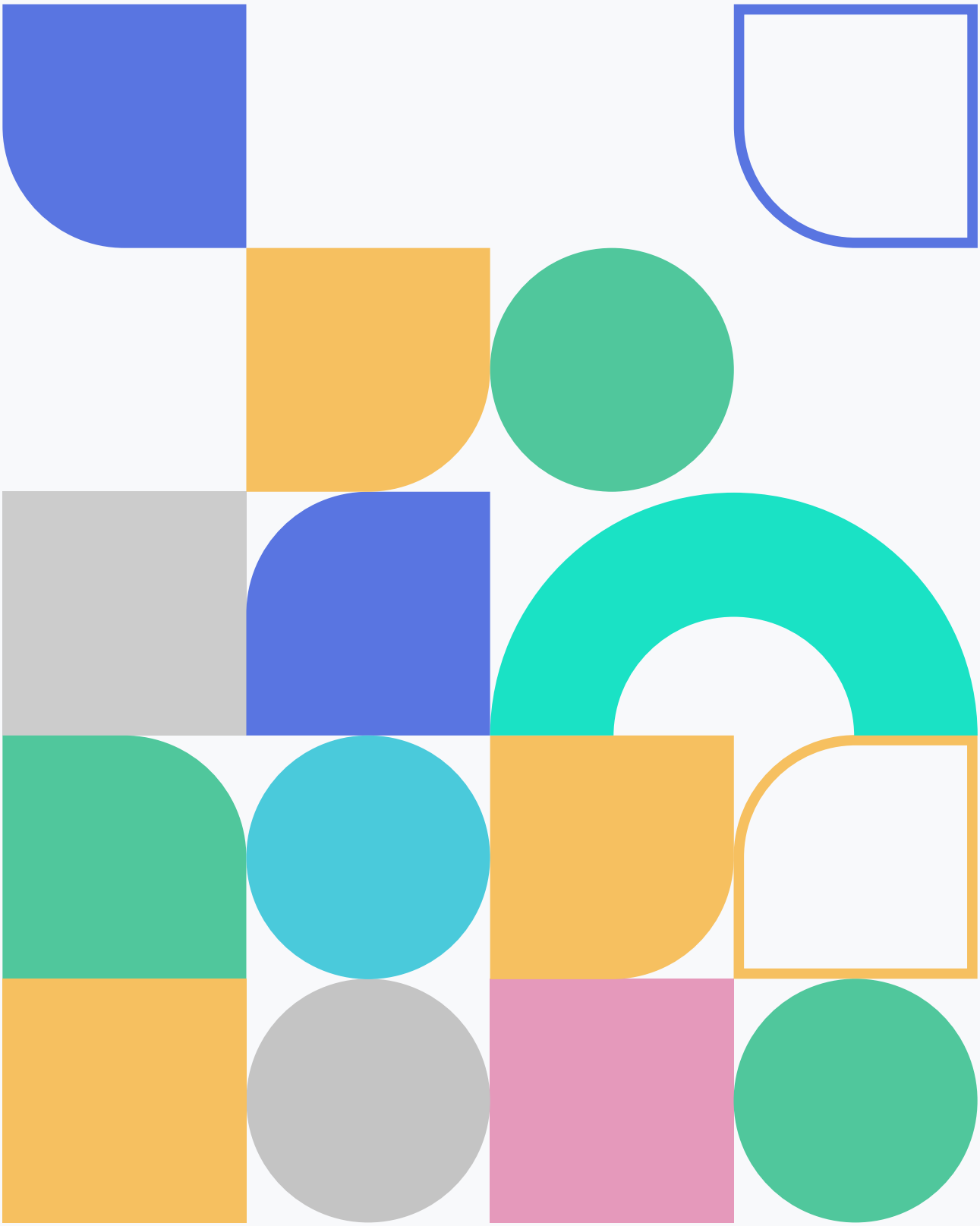
Schließlich fanden alle Probanden den Kalender schnell und mühelos. Die Monatsansicht ist bereits bekannt. Die ausgegraute Felder in der Woche, die zum vorherigen oder nächsten Monat gehören, haben die Orientierung erleichtert. Auch die gerade getätigten Einträge wurden am aktuellen Tag gefunden und

erkannt. Was allerdings eine andere Anordnung benötigt, ist die Navigation am unteren Bildschirmrand. Das Dashboard ist der Wichtigkeit wegen neben dem größten Button angeordnet, einige Probanden wünschen sich dies allerdings auf der linken Seite. Auch das Pluszeichen zum Hinzufügen von Einträgen ist von der Verständlichkeit her nicht optimal. Hier müssten Alternativen ausprobiert und entsprechende Tests durchgeführt werden.

Allgemein kamen Vorschläge wie Vorausplanungen besonders in Bezug auf das Modul Bewegung sowie Erinnerungen und Benachrichtigungen auf das Handy. Über letzteres wurde im Team bereits heiß diskutiert. Wie solche Pushbenachrichtigungen aussehen könnten, sodass sie den freundlichen Charakter der App wahren, wurde bereits im letzten Semester besprochen. Auch das Eintragen von Arztterminen wurde vorgeschlagen, was bereits in unserer Konzeption angedacht ist. Hier finden wir also Bestätigung für eine Notwendigkeit in der Implementierung. Von allen Probanden war zu hören, dass das Design simpel und modern gehalten ist. Die Screens wirken nicht überladen sowie professionell und sind bis auf die zuvor genannten Anordnungsprobleme übersichtlich und ruhig gehalten. Auch ist die Anwendung individuell genug mit den sechs geplanten Modulen, von denen zwei bisher umgesetzt sind. Die Probanden würden die App auch gerne für sich selbst nutzen, um z. B. Migräne oder Nackenschmerzen zu tracken.

Diese generelle Richtung bedeutet für das Projekt fine, dass wir auf dem richtigen Weg sind, aber dennoch mit einigen Baustellen zu rechnen ist, die ein neues Design, eine neue Umsetzung oder gar einige Feature-Verbesserungen mit sich bringen. Dies sind gewiss gute erste Einblicke in die User Experience von fine. So steht für dieses Projekt als nächstes Iterationen an User Tests und Verbesserungen an, um die Sache wirklich rund abzuschließen.

11. LIVEGANG



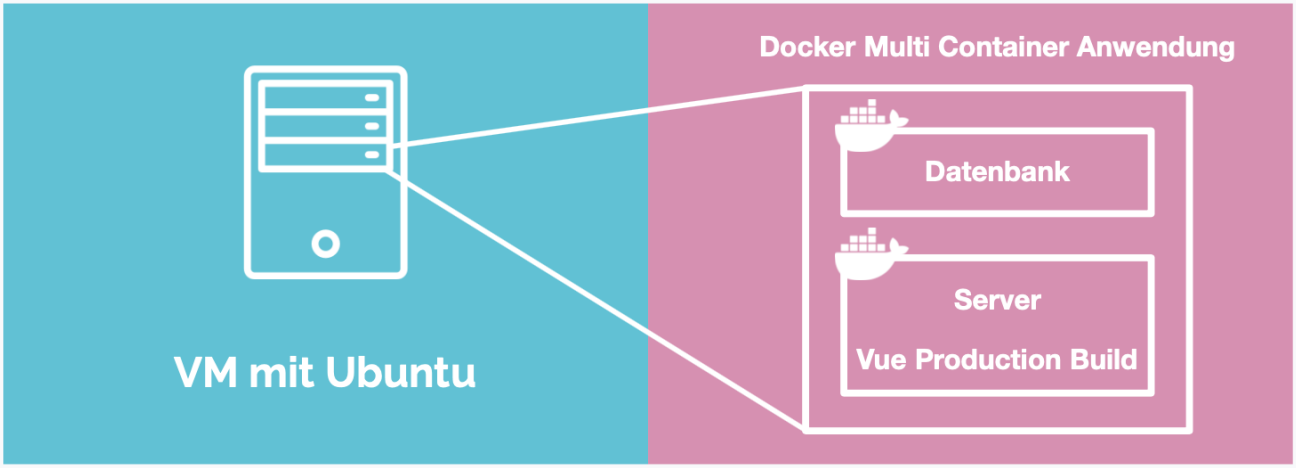


Abbildung 75: Ubuntu Server mit Docker-Container

Wir sind mit unserer App live gegangen. Dafür haben wir die von Prof. Dr. Wolfgang Kowarschick und Stefan König zu Verfügung gestellte Virtuelle Maschine (VM) genutzt. Diese lässt sich im Hochschulnetz unter fine-app.multimedia.hs-augsburg.de erreichen.

Auf dieser VM ist Ubuntu installiert. Ein Apache Webserver dient als Proxy, sodass nur auf gewisse Ports zugegriffen werden kann. Dies ist notwendig, da Docker standardmäßig alle Ports nach außen öffnet und so Sicherheitsrisiken entstehen können.

Nachdem wir unser Backend inklusive einem Production-Build des Vue-Frontends auf die VM kopiert haben, mussten wir nur noch Docker Engine und Docker Compose installieren. Nachdem wir den Proxy des Apache Webservers angepasst haben, konnten wir unsere Anwendung ganz einfach per Command Line mit Docker Compose starten.

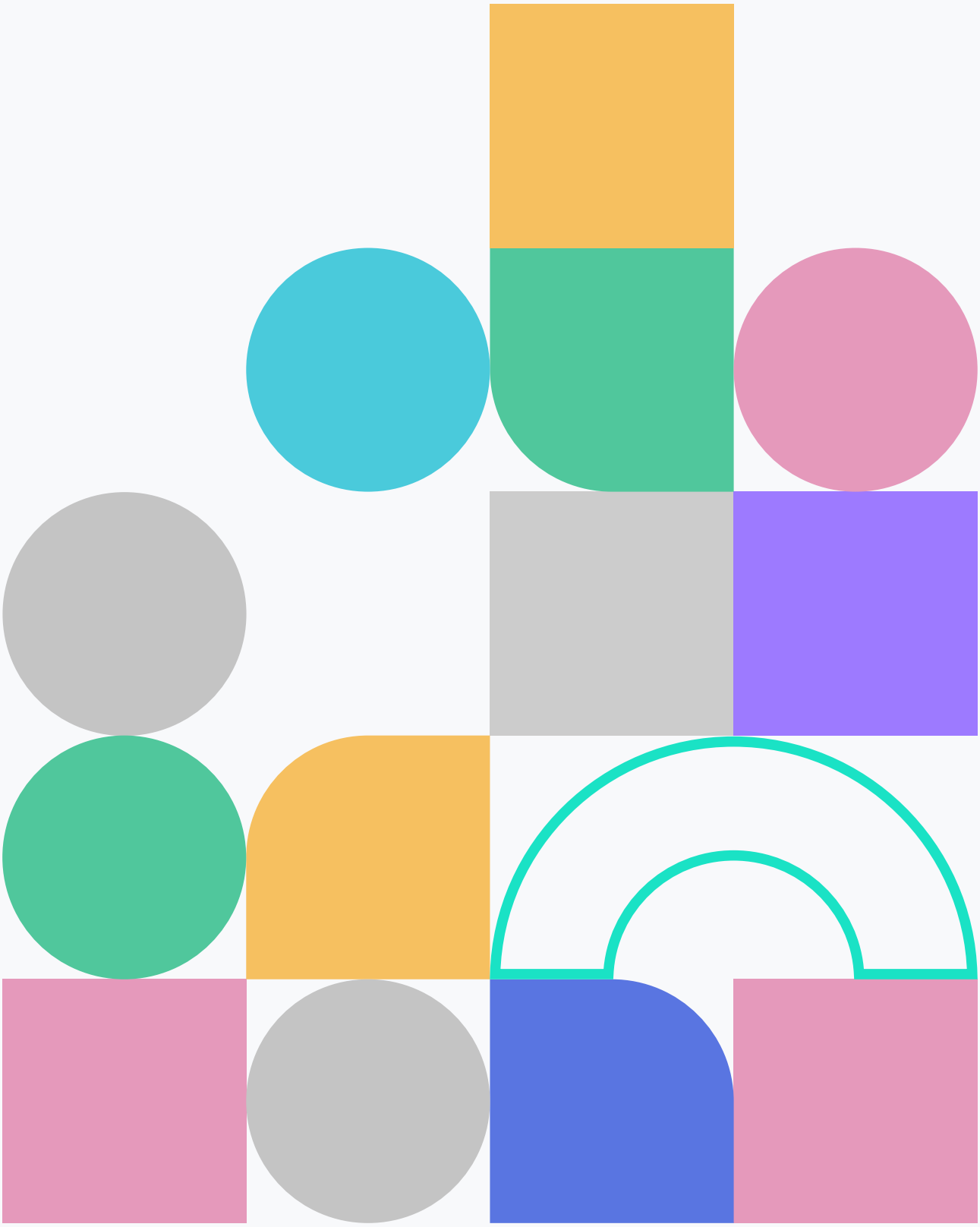
12. RESÜMEE UND AUSBLICK

Fine hat die zu Beginn des Semesters gesetzten Ziele übertroffen und ist auf dem Weg zu einem guten lauffähigen Prototypen. Nicht nur das Modul Symptome wurde umgesetzt, sondern zusätzlich auch Gefühle. Eine weitere Umsetzung von Modulen wird sich somit immer einfacher gestalten, da bereits entwickelte Komponenten als Vorlage dienen. Auch Kalender, Dashboard, User Login sowie die Verknüpfung der Daten mit dem Backend sind zustande gekommen. So steht einer Funktionserweiterung nichts im Weg.

Im Design wurden alle Module in Smartphone- und Tabletgröße umgesetzt. Hier gilt es, Verbesserungen durchzuführen, nachdem Nutzertests gemacht wurden. Für die Statistik müssen erste Konzeptideen aus dem Konzeptionssemester aufgegriffen und bis in die Tiefe durchdesignt werden, um sie dann im Prototypen zu integrieren. Kommen erst einmal alle Module sowie die Statistik in die Entwicklung, wird der erste Mehrwert der App zum Vorschein kommen.

Dank des bereits erstellten Brandings wird die Präsentation der eigenen Marke auf dem Markt kein Problem darstellen. Was eher noch besser integriert werden muss, ist der Datenschutz. Das angedachte Konzept aus dem Ethikkurs muss hier weiter ausdiskutiert und umgesetzt werden. Auch gilt es, Langzeittests mit Nutzern durchzuführen, die die App für mehrere Wochen oder Monate ausprobieren. Mit diesen Ergebnissen können Verbesserungen, wo notwendig, vorgenommen werden. Erste Interviews mit Nutzern zeigen, dass unsere App Potenzial hat. Dies würde bedeuten, dass die Menschen von unserem Tagebuch profitieren und sich und ihre Krankheit besser kennenlernen könnten. Genau das würden wir uns sehr für die Menschen wünschen, da der aktuelle Markt sich nicht darauf fokussiert.

13. ANHANG



13.1. QUELLEN

[1] GitLab. (12. Juli 2021). Simplify your Workflow with GitLab. https://about.gitlab.com/stages-devops-lifecycle/

[2] Docker. (12. Juli 2021). What is a Container?. https://www.docker.com/resources/what-container

[3] Docker. (12. Juli 2021). Overview of Docker Compose. https://docs.docker.com/compose/

[4] ESLint. (13. Juli 2021). Getting Started with ESLint. https://eslint.org/docs/user-guide/getting-started

[5] MongoDB. (13. Juli 2021). Introduction to MongoDB. https://docs.mongodb.com/manual/introduction/

[6] DB Engines. (13. Juli 2021). Vergleich der Systemeigenschaften MongoDB vs. PostgreSQL. https://db-engines.com/de/system/MongoDB%3BPostgreSQL

[7] Nodejs. (13. Juli 2021). Über Node.js. https://nodejs.org/de/about

[8] npm. (13. Juli 2021). About npm. https://www.npmjs.com/about

[9] Express.js. (13. Juli 2021). https://expressjs.com/de/

[10] Kalkuhl, L. (27.08.2020) GraphQL im Vergleich mit REST. doubleslash. https://blog.doubleslash.de/graphql-im-vergleich-mit-rest/

[11] Joi. (13.07.2021) Introduction. https://joi.dev/

[12] Filipova, O. (2016) Learning Vue.js 2 : learn how to build amazing and complex reactive web applications easily with Vue.js.

[13] Vue. (13. Juli 2021). Introduction. https://v3.vuejs.org/guide/introduction.html

[14] Filipova, O. (2016) Learning Vue.js 2 : learn how to build amazing and complex reactive web applications easily with Vue.js.

[15] Vue. (13. Juli 2021). Introduction. https://v3.vuejs.org/guide/introduction.html

[16] Vuex. (13. Juli 2021). What is Vuex. https://next.vuex.vuejs.org/

[17] MDN Web Docs. (13. Juli 2021). Window.localStorage. https://developer.mozilla.org/de/docs/Web/API/Window/localStorage

[18] Sass. (20. Juli 2021). Index. https://sass-lang.com/

[19] css-tricks. (20. Juli 2021). Let's Make a Vue-Powered Monthly Calendar. https://css-tricks.com/lets-make-a-vue-powered-monthly-calendar/

[20] dayjs. (20. Juli 2021). Index. https://day.js.org/

[21] Github. (20. Juli 2021). Mitt Repository. https://github.com/developit/mitt

[22] Vuejs. (20. Juli 2021). Vue 3 Migration Guide. https://v3.vuejs.org/guide/migration/events-api.html#root-component-events

[23] npm. (20. Juli 2021). npm svg-sprite package. https://www.npmjs.com/package//svg-sprite

[24] Mozilla. (13. Juli 2021). SpeechRecognition. https://developer.mozilla.org/en-US/docs/Web/API/SpeechRecognition#browser_compatibility

[25] Shires, G. (13. Juli 2021). Voice Driven Web Apps: Introduction to the Web Speech API. Google. https://developers.google.com/web/updates/2013/01/Voice-Driven-Web-Apps-Introduction-to-the-Web-Speech-API

[26] Mozilla. (13. Juli 2021). Using the Web Speech API. https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API/Using_the_Web_Speech_API

[27] W3C Community Group. (13. Juli 2021). Web Speech API. https://wicg.github.io/speech-api/

[28] Github. (13. Juli 2021). Pocketsphinx.js. https://syl22-00.github.io/pocketsphinx.js/

[29] Google. (13. Juli 2021). Cloud Speech-to-Text API. https://cloud.google.com/speech-to-text/docs/reference/rest?hl=de

13.2. ABBILDUNGSVERZEICHNIS

13.3. ABKÜRZUNGSVERZEICHNIS

DOM	Document Object Model
JWT	JSON Web Tokens
MVVM	Model-View-View-Model
VDOM	Virtual Document Object Model

13.4. QUELLCODEVERZEICHNIS