



“ Der Ostwestfale an sich ramentert gerne. Hat er sich beruhigt, drömelt er zuhause rum – am liebsten im Pölter und mit Schluffen an den Füßen. Es sind diese Wörter und Redensarten, die typisch für die ostwestfälische Mundart sind. ”

– Lippische Landes-Zeitung, 2013: Die Unvergänglichkeit des Ostwestfälischen.

I Inhaltsverzeichnis

I	Inhaltsverzeichnis	II
II	Abbildungsverzeichnis	III
1	Einleitung & Zielsetzung	1
1.1	Vorgeschichte & Motivation	1
1.2	Zielsetzung	1
1.3	Aufbau der Arbeit	2
2	Idee & Konzept der App	3
2.1	Entstehung der Idee	3
2.2	Features & Funktionen	4
2.2.1	Das Quiz	4
2.2.2	Das Wörterbuch	7
2.2.3	Begriff des Tages	8
2.2.4	Die Statistik	9
2.2.5	Die Hilfe	9
2.2.6	Wörterbuch ändern	9
2.3	Layout & Navigation	10
2.3.1	Status Bar	11
2.3.2	Navigation Bar	12
2.3.3	View	13
2.3.4	Tab Bar	13
2.4	Didaktisches Konzept	15
2.4.1	Behaviorismus	15
2.4.2	Kognitivismus	17
2.5	Ostwestfälische Begriffe	20
3	Gestaltung der App	21
3.1	Überblick & Screenshots	21
3.2	Farbgestaltung	24
3.3	Typografie & Icons	26
3.4	Gestaltungselemente	30
3.4.1	Tab Bar	30
3.4.2	Navigation Bar	31
3.4.3	Titel & Texte	33
3.4.4	Buttons & Inputs	35
3.4.5	Listen	39
3.4.6	Diagramme	41
3.4.7	App-Icon	43
3.5	Stylesheet	44
4	Programmierung der App	48
4.1	Fremdcode & Frameworks	48
4.2	Module & Struktur	50
4.3	Daten & Dateien	53
4.4	Sonstige Hilfsmittel	56
5	Fazit	57
A	Literaturverzeichnis	A
B	Selbstständigkeitserklärung	B
C	Anhang	C

II Abbildungsverzeichnis

Abbildung 1	Wireframe: Quiz-Typ Begriff zu Bilder	5
Abbildung 2	Wireframe: Quiz-Typ Bild zu Begriffen	5
Abbildung 3	Wireframe: Quiz-Typ Begriff zu Buchstaben	6
Abbildung 4	Wireframe: Quiz-Typ Begriff zu Eingabe	6
Abbildung 5	Wireframe: Wörterbuch-Übersicht	7
Abbildung 6	Wireframe: Wörterbuch-Details	7
Abbildung 7	Wireframe: Begriff des Tages	8
Abbildung 8	Wireframe: Statistik des Nutzers	8
Abbildung 9	Wireframe: App-Layout	11
Abbildung 10	Flussdiagramm zum Behaviorismus in der App	16
Abbildung 11	Screenshots: Begriff des Tages & Statistik	21
Abbildung 12	Screenshots: Quiz-Ende & -Typen	22
Abbildung 13	Screenshots: Mehr, Hilfe & Optionen	23
Abbildung 14	Screenshots: Wörterbuch-Übersicht & -Details	24
Abbildung 15	Farbpalette der App	25
Abbildung 16	Schriftart Lato	26
Abbildung 17	Übersicht der Icons in der App	28
Abbildung 18	Detailansicht: Tab Bar	30
Abbildung 19	Detailansicht: Navigation Bar	32
Abbildung 20	Detailansicht: Titel & Texte	34
Abbildung 21	Detailansicht: Buttons	36
Abbildung 22	Detailansicht: Inputs	38
Abbildung 23	Detailansicht: Listen	40
Abbildung 24	Detailansicht: Diagramme	42
Abbildung 25	App-Icon von OWLisch	43
Abbildung 26	Flussdiagramm zum Mediator-Pattern	51

1 Einleitung & Zielsetzung

Im nachfolgenden werden die Motivation, Zielsetzung und der Aufbau der vorliegenden Bachelorarbeit erläutert.

1.1 Vorgeschichte & Motivation

Im Zuge des Seminars *Mobile Learning* im Wintersemester 2015/16 an der Universität Bielefeld unter Leitung von Paul John war ich, zusammen mit sieben weiteren Studierenden, an der Konzeption, Gestaltung und Umsetzung einer Lern-App für Dialektwörter aus Ostwestfalen-Lippe beteiligt. Dabei entstand ein erster funktionsfähiger Prototyp der App *OWLisch*, an dessen Programmierung ich maßgeblich mitgewirkt habe.

Bereits beim Entwickeln dieser ersten rudimentären Version habe ich einen tieferen Einblick sowohl in die Programmierung von Web-Apps mit der Programmiersprache JavaScript, als auch in die didaktische Konzeption einer solchen Lern-App erhalten. Mit dieser Arbeit wollte ich noch einen Schritt weiter gehen und aus der bereits bestehenden Idee ein fertiges Produkt erstellen, das auf aktuellen Technologien basiert und mein Verständnis für die damit verbundenen Programmiersprachen und Konzepte vertieft.

1.2 Zielsetzung

Das Ziel dieser Arbeit war es, eine lauffähige iOS-Version von *OWLisch* zu entwickeln. Dabei sollte die App von Grund auf neu programmiert werden; ich entschied mich allerdings dazu, die App nicht in der von Apple kreierten Programmiersprache *Swift* zu entwickeln, sondern einen Hybrid-Ansatz zu wählen, um größtmögliche Flexibilität zu erreichen.

Dabei wollte ich auf etablierte Web-Technologien wie JavaScript, HTML und CSS zurückgreifen und diese mit dem Framework *Apache Cordova* kombinieren. Dies erlaubt es, eine so erstellte App nicht nur in jedem modernen Web-Browser auszuführen, sondern diese auch mit geringem Aufwand zu einer nativen iOS- oder sogar Android-App zu konvertieren, ohne den Programmcode für jede Plattform neu schreiben zu müssen.

Der eigentliche Inhalt und das Konzept des bestehenden Prototyps sollten dabei nur geringfügig verändert oder erweitert werden; die in dieser Arbeit enthaltene Beschreibung der Konzeption und Gestaltung stimmen daher größtenteils mit der Vorgängerversion überein.

1.3 **Aufbau der Arbeit**

Diese Arbeit beschäftigt sich zunächst mit der grundlegenden Idee, der Funktionsweise, dem Aufbau und dem zugrunde liegenden didaktischen Konzept der App *OWLisch*. Nachfolgend wird die grafische Gestaltung der App näher beleuchtet und im Detail beschrieben, wobei ein Vergleich zu den *iOS Human Interface Guidelines* gezogen und eine erste Verbindung zum dazugehörigen Quellcode hergestellt wird.

Im nächsten Kapitel des Hauptteils wird die eigentliche Programmierung der App näher beschrieben; dabei werden die eingesetzten Technologien und Bibliotheken, der modulare Aufbau des Programms und die verwendeten Programmiermuster in der Sprache *JavaScript* präsentiert. Das letzte Kapitel fasst die Arbeit zusammen und gibt einen Ausblick auf eine mögliche Fortsetzung des Projekts.

2 Idee & Konzept der App

Nachfolgend wird die Idee der App, die Funktionsweise und das dahinter liegende didaktische Konzept erläutert.

2.1 Entstehung der Idee

Die Idee für die App *OWLisch* entstand im Zuge des Seminars *Mobile Learning* unter Beteiligung folgender Studierenden: Tugba Aksakal, Miriam Belke, Melanie Derksen, Franziska Kluge, Lisa Kottmann, Kai-Frederik Lüking, Jakob Metzger und Philipp Niewöhner. Ziel war es zunächst, ein Konzept für eine Lern-App für Dialektwörter aus Ostwestfalen-Lippe unter Berücksichtigung von didaktischen Erkenntnissen zu entwickeln; zu diesem Zweck wurden während des Seminars interdisziplinäre Gruppen aus verschiedenen Studiengängen gebildet (Medienwissenschaften, Erziehungswissenschaften und Medieninformatik & Gestaltung).

Als Inspiration für die App dienten in erster Linie die bekannte Spiele-App *Quizduell* und das althergebrachte Karteikartensystem zum Lernen von Vokabeln. Bei mehreren Gruppendiskussionen wurden diese Beiden Konzepte vereint und konkretisiert.

Um das Lernen der ostwestfälischen Vokabeln aufzulockern, wurde ein sogenannter Gamification-Ansatz gewählt – das Anwenden spieltypischer Elemente in einem sonst spielfremden Kontext – in Anlehnung an die App *Quizduell*. Die in der App enthaltenen Begriffe sollten als ein Quiz präsentiert werden, das über mehrere Schwierigkeitsstufen verfügt und den Spieler bei richtigen Antworten mit einem kleinen Erfolg belohnt.

Im Hintergrund, für den Spieler weitestgehend unsichtbar, sollte ein einfaches Karteikartensystem implementiert werden; Begriffe, die falsch erraten werden, sollten häufiger im Quiz auftauchen und so in den Vordergrund rücken, bis der Spieler die entsprechende Frage richtig beantwortet.

2.2 Features & Funktionen

Nachdem die Kernidee der App ausformuliert wurde, konnten konkrete Funktionalitäten erdacht und skizziert werden. Im Fokus stand dabei zunächst das Quiz als Hauptbestandteil der App; dieses sollte über mehrere Quiz-Typen verfügen, die verschiedene Schwierigkeitsgrade darstellen und dem Spieler sowohl Abwechslung, als auch eine Herausforderung bieten.

2.2.1 Das Quiz

In der ersten und leichtesten Stufe wird dem Spieler zunächst eine simple Multiple-Choice-Frage über einen Begriff mit vier Antwortmöglichkeiten präsentiert. Dies ermöglicht es dem Spieler, durch einfaches Raten auf die richtige Lösung zu kommen. Dieser Typ teilt sich wiederum in vier Sub-Typen auf:

- Ein ostwestfälischer Begriff mit vier hochdeutschen Begriffen als Antwortmöglichkeit (z.B. „Was heißt Pinneken?“).
- Ein hochdeutscher Begriff mit vier ostwestfälischen Begriffen als Antwortmöglichkeit (z.B. „Was heißt Schnapsglas?“).
- Ein ostwestfälischer Begriff mit vier Bildern als Antwortmöglichkeiten (z.B. „Was heißt Schnapsglas?“) [siehe Abbildung 1](#).
- Ein Bild zu einem Begriff mit vier ostwestfälischen Begriffen als Antworten [siehe Abbildung 2](#).

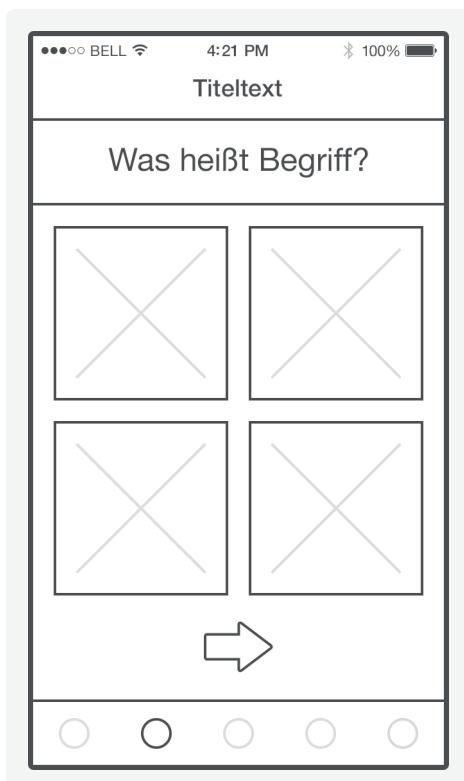


Abbildung 1

Wireframe: Quiz-Typ
Begriff zu Bilder

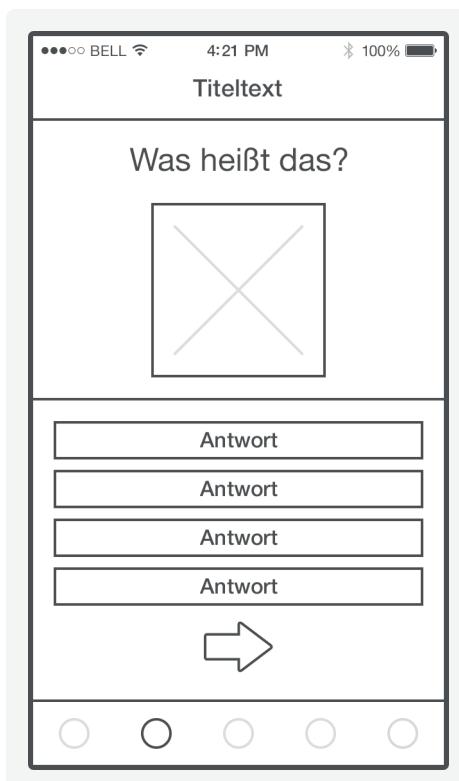


Abbildung 2

Wireframe: Quiz-Typ
Bild zu Begriffen

Diese vier Typen sollten dabei jedes Mal zufällig ausgewählt werden. Die Bilderrätsel lockern dabei einerseits die sonst stark textbasierte App auf und unterstützen andererseits visuelles Lernen und können dem Spieler so eine zusätzliche Erinnerungsstütze bieten, wenn der entsprechende Begriff noch einmal im Quiz auftaucht.

Wird ein Begriff während dieser ersten Stufe richtig erraten, steigt der Spieler für diesen um eine Stufe auf und erhält beim nächsten Vorkommen einen neuen Schwierigkeitsgrad. Bei dieser zweiten Stufe wird dem Spieler die hochdeutsche Übersetzung des Begriffs präsentiert, während alle im ostwestfälischen Wort vorkommenden Buchstaben zufällig gemischt dargestellt sind und in die richtige Reihenfolge gebracht werden müssen.

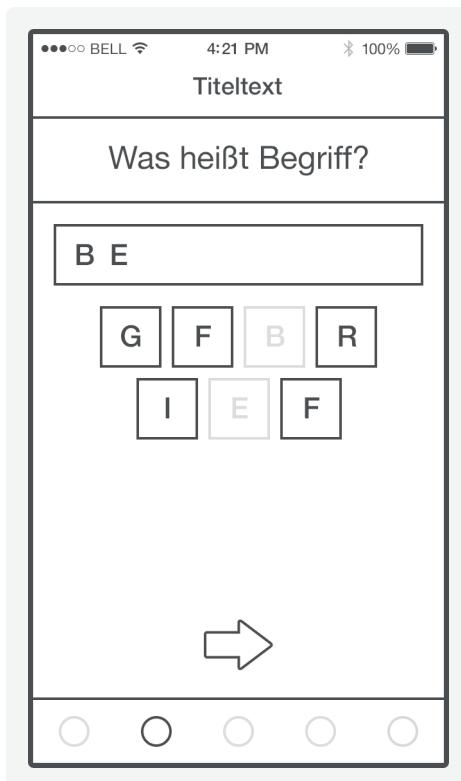


Abbildung 3

Wireframe: Quiz-Typ
Begriff zu Buchstaben

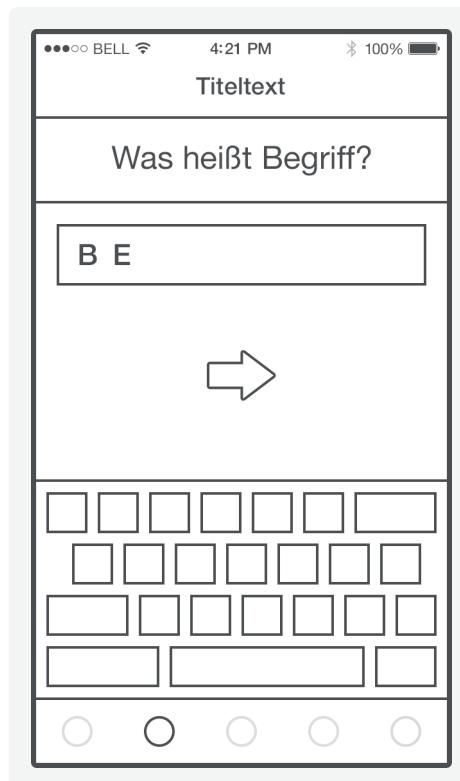


Abbildung 4

Wireframe: Quiz-Typ
Begriff zu Eingabe

Hierdurch soll der Spieler die korrekte Schreibweise des Begriffes erlernen, erhält durch die Vorgabe der Buchstaben allerdings weiterhin eine Hilfestellung, sodass die Lösung durch Knobeln und ausprobieren erreicht werden kann. Über einen vorhandenen „Lösen“-Button kann die Eingabe bestätigt werden [siehe Abbildung 3](#).

Die letzte und somit schwierigste Stufe ähnelt ihrem Vorgänger; auch hier wird die hochdeutsche Übersetzung dargestellt, es entfällt jedoch die Hilfestellung. Der Spieler muss den ostwestfälischen Begriff selbstständig mittels einer eingebetteten Tastatur korrekt eingeben. Kommt ein Spieler hier auf die richtige Lösung, so wurde der Begriff erfolgreich erlernt [siehe Abbildung 4](#).

2.2.2 Das Wörterbuch

Ein weiteres geplantes Feature für die App ist das persönliche Wörterbuch des Spielers. Hier sollen automatisch alle Begriffe hinzugefügt werden, die im Quiz richtig erraten wurden; so kann der Nutzer bereits kennengelernte Begriffe erneut nachschlagen und zusätzliche Information darüber erhalten.

Das Wörterbuch verfügt sowohl über eine Übersichtsliste aller Begriffe [siehe Abbildung 5](#), als auch über Detailansichten, die vorhandene Bilder und Herkünfte, Zitate und Übersetzungen über den Begriff enthalten [siehe Abbildung 6](#). Zusätzlich soll hier eine auditive Komponente ergänzt werden, indem alle Begriffe als eingesprochene Audiodateien im Wörterbuch abgespielt werden können, sodass der Nutzer die korrekte Aussprache der Wörter lernen kann.

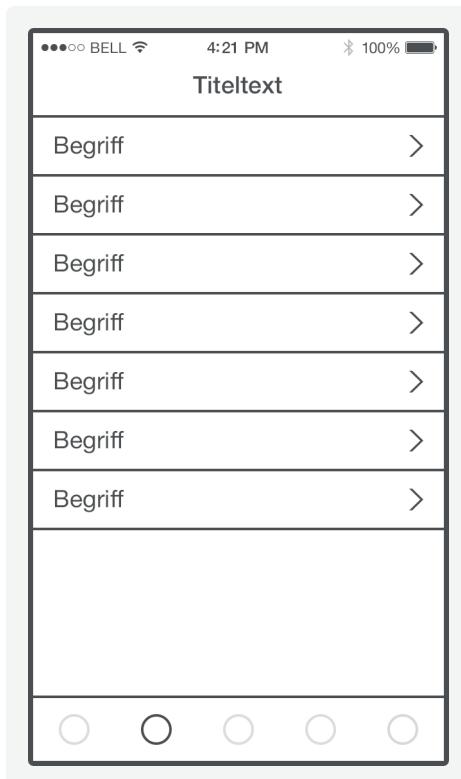


Abbildung 5

Wireframe: Wörterbuch-Übersicht

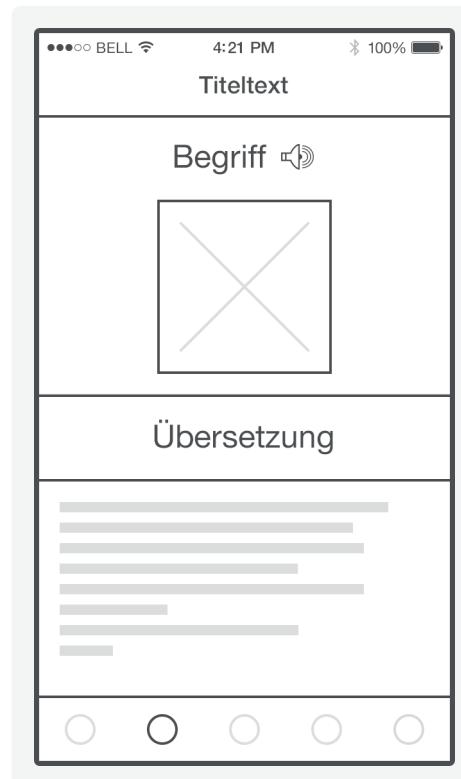


Abbildung 6

Wireframe: Wörterbuch-Details

2.2.3 Begriff des Tages

Um dem Nutzer beim Starten der App einen ersten Eindruck über die Inhalte zu verschaffen und nicht sofort in einen der Menüpunkte zu werfen, kam die Überlegung auf, den *Begriff des Tages* als Startbildschirm einzusetzen.

Dieser Bildschirm präsentiert dem Nutzer einen der in der App vorhandenen Begriffe mitsamt Bild, Übersetzung und Informationstext (ähnlich wie bei den Wörterbuch-Details), sodass man bereits vor dem ersten Spielen des Quiz ein neues Wort kennenlernen kann [siehe Abbildung 7](#). Dieser Begriff wird täglich zufällig gewechselt und kann dem Nutzer somit sowohl bereits bekannt, als auch völlig neu sein. Gelernte Begriffe können so noch einmal aufgefrischt werden, sodass sie beim nächsten Spiel einfacher zu lösen sind.

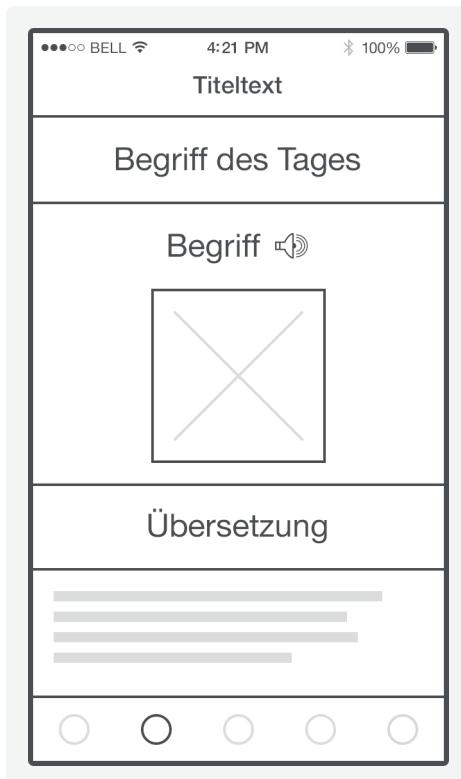


Abbildung 7

Wireframe: Begriff des Tages

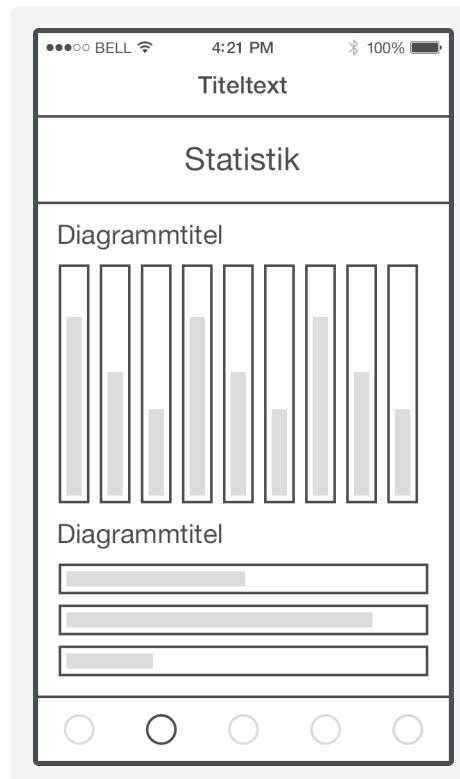


Abbildung 8

Wireframe: Statistik des Nutzers

2.2.4 Die Statistik

Damit der Nutzer der App zu jeder Zeit einen Überblick über den bisher erzielten Lernfortschritt behält, wurde beschlossen, eine persönliche Statistik zu implementieren, in der über Diagramme wichtige Informationen dargestellt werden [siehe Abbildung 8](#).

Hier erhält man Einsicht in die erzielten Punkte aus den letzten Quiz-Spielen, die Anzahl der bisher gelernten Begriffe in Relation zur Gesamtzahl der Begriffe und eine Aufschlüsselung der aktuellen Stufen der gelernten Begriffe. Auf diese Weise soll dem Nutzer der Fortschritt visuell vor Augen geführt werden, sodass ersichtlich ist, was bereits erreicht wurde und wie viel noch fehlt, bis alle Begriffe erfolgreich erlernt wurden.

2.2.5 Die Hilfe

Da trotz sorgfältiger Überlegungen hinsichtlich der Inhalte und Funktionsweisen der App beim Nutzer dennoch Fragen zur Bedienung aufkommen können, sollte zusätzlich noch ein ausführlicher Hilfetext in der App untergebracht werden. Hier sollen die wesentlichen Funktionen vom Quiz, dem Wörterbuch und der Statistik verständlich erklärt werden.

2.2.6 Wörterbuch ändern

Es kam bereits während der Planungsphase die Idee auf, die App nicht nur auf Ostwestfälische Begriffe zu beschränken, sondern auch andere deutsche Dialekte zu unterstützen. Dieses Feature wurde zwar bei der Entwicklung des ersten Prototypen verworfen, bei der finalen Umsetzung jedoch wieder aufgegriffen; näheres dazu findet sich in *Kapitel 3* und *Kapitel 4*.

2.3 Layout & Navigation

Nachdem geklärt war, welche Grundfeatures die App enthalten soll und wie diese in etwa gestaltet sein sollten, musste das Konzept noch um eine Navigation und Inhaltsstruktur erweitert werden. Aufgrund der vorhandenen persönlichen Geräte und der bereits gemachten Erfahrungen der Gruppenmitgliedern mit unterschiedlichen Smartphones und Betriebssystemen, wurde iOS als Zielplattform für den ersten Prototypen ausgewählt, mit dem iPhone 5 (und später dem iPhone SE) als Testgerät.

Als primäre Quelle für das grundlegende Layout und die Navigationsstruktur der App wurden die *iOS Human Interface Guidelines*¹ herangezogen; hier sind sämtliche von Apple empfohlenen Interface-Elemente für Apps dokumentiert und beschrieben. Das Einhalten von Richtlinien und Empfehlungen ist beim Entwerfen einer App von erheblicher Bedeutung, um dem Endnutzer eine intuitive Bedienung der App zu ermöglichen und sich auf den Inhalt zu konzentrieren. Auch andere beliebte Apps wie Facebook und WhatsApp dienten der Gruppe als Inspiration für die Gestaltung des Layouts.

Für die Inhaltsstruktur wurde dabei eine flache Hierarchie gewählt, die mit minimalen Verschachtelungen und möglichst wenigen Menüs auskommt, damit der Nutzer stets die Übersicht behält, wo er sich gerade in der App befindet. Die verwendeten Layout- und Navigationselemente und ihre Beziehung zu den zuvor beschriebenen Features werden im folgenden im Detail beschrieben und bildlich dargestellt *siehe Abbildung 9*, auf die finale Gestaltung dieser Elemente wird in Kapitel 3 eingegangen.

1. Vgl. Apple Inc., 2016: *iOS Human Interface Guidelines*.
<https://developer.apple.com/ios/human-interface-guidelines> (15.12.2016)

2.3.1 Status Bar

Die *Status Bar* ist in iOS-Apps ein statischer und weitestgehend unveränderlicher Bestandteil, der sich stets am oberen Rand des Bildschirms befindet und wichtige Systeminformationen wie die Netzwerk-Konnektivität, die Uhrzeit und den aktuellen Batterie-Ladestand darstellt.

Dabei ist der Inhalt dieser Leiste global gültig und damit unabhängig von der gerade geöffneten App. Die einzige Ausnahme, die die *iOS Human Interface Guidelines* erlauben, liegt bei Spiele-Apps vor, die im Vollbildschirm-Modus betrieben werden; hier kann es sinnvoll sein, die *Status Bar* vollständig auszublenden, was bei *OWLisch* allerdings nicht zutrifft. Daher muss beim Gestalten des App-Layouts Platz für diese Leiste eingeräumt werden.

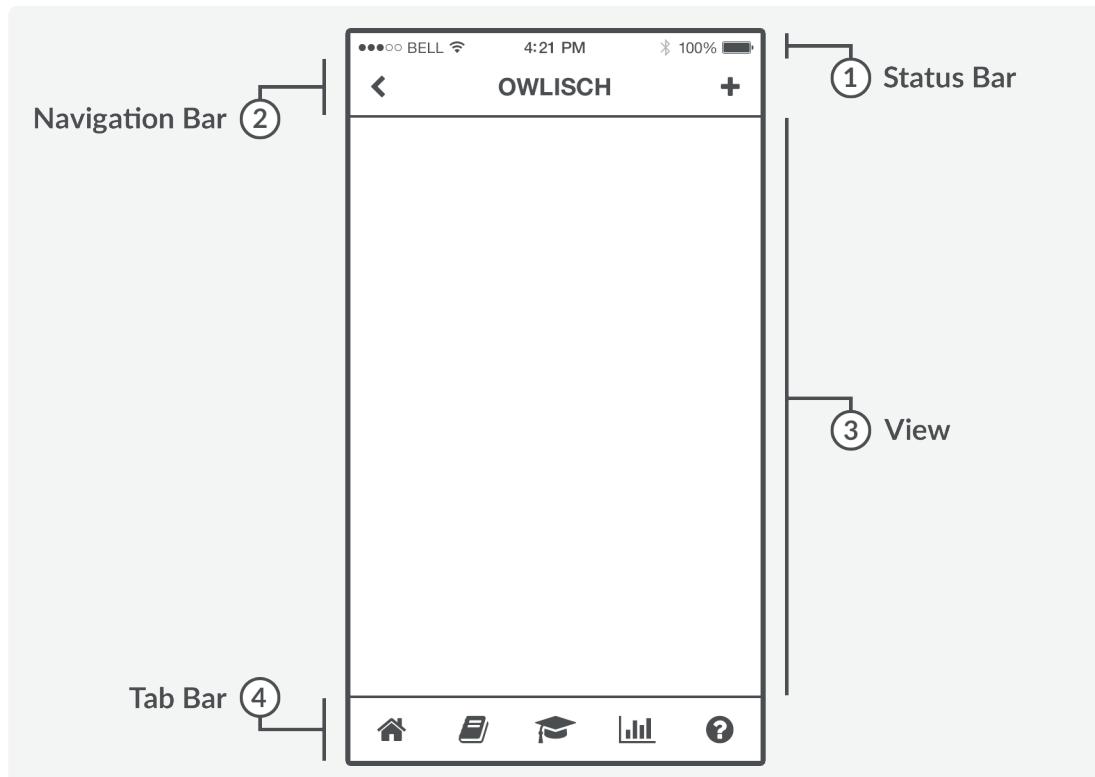


Abbildung 9

Wireframe: App-Layout

2.3.2 Navigation Bar

Die *Navigation Bar* ist in iOS von wesentlicher Bedeutung für die Navigation innerhalb der App. In der Regel ist diese Leiste visuell mit der zuvor erwähnten *Status Bar* verknüpft und enthält kontextabhängige Informationen über den aktuellen Inhalt der App. Hier kann beispielsweise der Titel der gerade gewählten View [siehe 2.3.3](#) angezeigt werden. Auch ist es üblich und empfohlen, auf der linken Seite der Leiste einen Zurück-Button (z.B. als Pfeilsymbol) zu platzieren, um innerhalb des aktuellen Kontextes zurück zu gelangen. Dies trifft bei OWLisch vor allem auf das Wörterbuch zu, innerhalb dessen man sich zwischen der Begriff-Liste und den Begriff-Details vor und zurück navigieren können soll [siehe 2.2.2](#).

Des Weiteren lassen sich in der *Navigation Bar* auch zusätzliche Buttons unterbringen, um kontextabhängige Aktionen auszuführen; im Wörterbuch könnten hier Buttons in Form von Symbolen sein, um die Begriff-Liste zu sortieren oder mittels einer Suchfunktion zu filtern, während im Quiz ein Abbrechen-Button platziert werden kann, mit dem das bereits gestartete Quiz beendet werden kann.

Die Platzierung der *Navigation Bar* ist statisch, sie kann also nicht beliebig verschoben werden; auch sollte sie zu jeder Zeit sichtbar sein, um dem Nutzer stets die Übersicht über seine aktuelle Position innerhalb der App zu gewähren. Für Apps, die einen großen Fokus auf die bloße Darstellung des Inhaltes legen, kann es sinnvoll sein, die *Navigation Bar* gelegentlich auszublenden, damit mehr Platz zur Verfügung steht (*Facebook* und *Safari* blenden die Leiste z.B. aus, wenn der Nutzer nach unten scrollt und wieder ein, sobald wieder nach oben gescrollt wird); für OWLisch ist dies jedoch nicht notwendig.

2.3.3 View

Die *View* ist das Herzstück der App, denn hier wird der eigentliche Inhalt angezeigt. Alle in *Kapitel 2.2* skizzierten Features werden hier platziert und abhängig vom gerade gewählten Menüpunkt [siehe 2.3.4](#) ein- und ausgeblendet. Dieses Element nimmt daher auch den größten Platz auf dem Bildschirm ein und lässt sich scrollen, wenn der dargestellte Inhalt über den vorhandenen Platz hinausgeht. Innerhalb der *View* kann der Nutzer unter Umständen ebenfalls navigieren, z.B. durch aus Auswählen eines Begriffes innerhalb der Begriff-Liste im Wörterbuch. Der Inhalt der zuvor beschriebenen *Navigation Bar* hängt maßgeblich davon ab, welcher Inhalt gerade in der *View* dargestellt wird. Im Grunde ist dieses Element nichts weiter als ein leerer Kasten und ist daher selbst in keiner Weise gestaltet.

2.3.4 Tab Bar

Die *Tab Bar* ist ein Standardelement aus den *iOS Human Interface Guidelines*, das in einer Vielzahl von Apps verwendet wird, darunter auch die meisten Apps von *Apple* selbst. Sie stellt die primäre Navigation für die App dar und befindet sich stets am unteren Rand des Bildschirms. In dieser Leiste können bis zu fünf Icons untergebracht werden, die einerseits eine Navigation zwischen den dargestellten Menüpunkten erlauben und andererseits durch die visuelle Hervorhebung des aktuell gewählten Icons stets anzeigen, wo man sich als Nutzer gerade befindet (ergänzend zum Titel in der *Navigation Bar*).

Dies gewährleistet die eingangs erwähnte flache Hierarchie der App; die fünf vorhandenen Menüpunkte liegen alle gleichberechtigt und ohne Verschachtelung auf der selben Navigations-Ebene. In *OWLisch* erhalten die fünf wichtigsten Features der App [siehe 2.2](#) jeweils ein eigenes Icon in der *Tab Bar*.

Das erste Icon verweist dabei auf den *Begriff des Tages*, danach folgen das *Wörterbuch*, das *Quiz*, die *Statistik* und schließlich die *Hilfe*. Falls weitere (weniger wichtige) Menüpunkte hinzukommen sollten, lässt sich der letzte Tab in der Leiste zum Menüpunkt „*Mehr*“ abwandeln, der dann eine Liste aus weiteren, diesem Punkt untergeordneten Optionen (z.B. die *Hilfe* oder das noch im Raum stehende Feature zum Ändern des Wörterbuchs) enthält. Diese Vorgehensweise wird auch von Apple vorgeschlagen, da so nicht nur für den Nutzer, sondern auch für den Entwickler während des Planungsprozesses klar wird, welche Funktionen der App am wichtigsten sind und daher am prominentesten dargestellt werden sollten.

Die *iOS Human Interface Guidelines* empfehlen außerdem, die *Tab Bar* stets anzuzeigen, unabhängig von der aktuellen Position innerhalb der Navigation, um dem Nutzer stets die Option zu bieten, zu jedem beliebigen Punkt zu navigieren. Gegen diese Regel wird in *OWLisch* beim Quiz allerdings bewusst verstößen werden; sobald ein Quiz gestartet wird, soll es dem Nutzer nämlich explizit nicht mehr erlaubt sein, einen anderen Menüpunkt auszuwählen, so lange das Quiz nicht beendet oder abgebrochen wurde. Dies soll verhindern, dass der Nutzer sich selbst betrügt, indem einfach im *Wörterbuch* oder beim *Begriff des Tages* nach der Lösung für eine Frage nachgesehen wird. Zudem gewährt das Ausblenden der *Tab Bar* unter Umständen mehr Raum, um die Inhalte des Quiz darstellen zu können und zusätzliches Scrollen seitens des Nutzer zu umgehen.

Für die Orientierung des Nutzers steht trotz ausgeblendeter *Tab Bar* weiterhin die *Navigation Bar* zu Verfügung, die anzeigt, dass man sich gerade im Quiz befindet und durch Buttons die Möglichkeit gibt, das Quiz jederzeit abzubrechen und damit die *Tab Bar* wieder einzublenden.

2.4 Didaktisches Konzept

Im nachfolgenden werden die didaktischen Prinzipien erläutert, die bei der Konzeption der App eine zentrale Rolle spielen. Dabei basiert das Lernkonzept von *OWLisch* hauptsächlich auf dem *Behaviorismus* und dem *Kognitivismus*, zwei psychologische Theorien, die in der Didaktik und im E-Learning Anwendung finden.

2.4.1 Behaviorismus

In der Lerntheorie des *Behaviorismus* steht das Verhalten der Lernenden im Vordergrund, während die internen psychologischen Prozesse außer Acht gelassen werden (sie sind eine *Black Box*). Das Lernen wird hierbei durch äußere Reize stimuliert, die bestimmte Reaktionen hervorrufen sollen. Gewollte positive Reaktionen werden durch Belohnungen bestärkt, während ungewollte Reaktionen durch Bestrafungen oder dem Ausbleiben von Belohnungen unterdrückt werden. Durch das Zusammenspiel von äußeren Reizen, der Reaktion des Lernenden und das anschließende Feedback wird eine *Reiz-Reaktions-Kette* aufgebaut, die einen objektiv messbaren Lernerfolg hervorrufen soll.²

Von zentraler Bedeutung für den Lernerfolg ist dabei eine klare Aufgabenstellung und ein eindeutiger, nachvollziehbarer Lösungsweg. Die einzelnen Schritte zum erzielten Lernerfolg und das Lernziel selbst müssen dem Lerndenen bekannt sein und in einer logischen Abfolge stattfinden. Durch eine kleinteilige Portionierung der Aufgaben kann eine schrittweise Annäherung an das Lernziel mit häufigen Belohnungen und einem stetig ansteigenden Schwierigkeitsgrad stattfinden.³

2. Vgl. Meir, Susanne, 2006: *E-Learning Plus*, Seite 10.

3. Vgl. Vontobel, Peter, 2006: *Didaktisches Design aus lernpsychologischer Sicht*, Seite 2-3.
Pädagogische Hochschule Zürich.

In OWLisch wird dieses Konzept hauptsächlich im Quiz realisiert; die gestellten Fragen dienen hier als Reiz, der beim Lernenden eine Reaktion in Form der Beantwortung der Frage hervorrufen soll. Wie der Lernende auf die richtige Antwort kommt, spielt dabei zunächst keine Rolle (die genannte *Black Box*). Auf die Reaktion folgt schließlich ein Feedback [siehe Abbildung 10](#) :

Wird die Frage richtig beantwortet, erhält der Nutzer eine Belohnung in Form eines Stufenanstiegs (dargestellt durch Sterne), einem neuen Eintrag im persönlichen Wörterbuch und dem Steigen der Fortschritts-Leiste in der Statistik-Ansicht. Bei einer falschen Lösung erfährt der Lernende zwar das richtige Ergebnis, eine Belohnung bleibt jedoch aus, um dieses ungewollte Verhalten nicht zu bestärken.

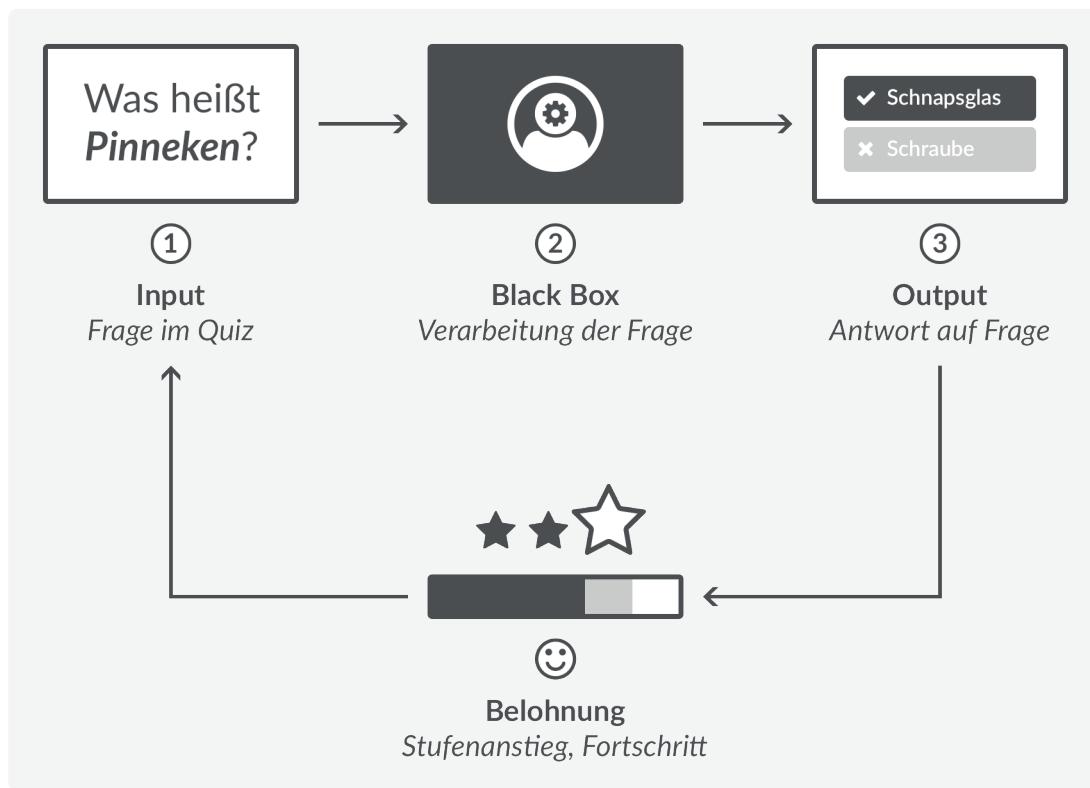


Abbildung 10

Flussdiagramm zum
Behaviorismus in der App

Durch die fehlende Belohnung wird der Lernende zudem dazu motiviert, die Aufgaben so lange zu wiederholen, bis sie korrekt gelöst wurden und so ein Erfolgserlebnis hervorrufen. Die Kleinteiligkeit der Aufgaben sorgt für häufige Erfolge und Belohnungen und folgt damit einem wichtigen Grundprinzip:

“*Verstärkung des Lernverhaltens durch Erfolg:
Wer erfolgreich lernt, lernt lieber und besser!*”

– Vontobel, Peter, 2006: Didaktisches Design aus lernpsychologischer Sicht,
Seite 3. Pädagogische Hochschule Zürich.

Durch die präzise Fragestellung und die Tatsache, dass bei jeder Frage eine eindeutige Lösung definiert ist, kann am Ende einer Quiz-Runde (bestehend aus zehn Fragen) ein objektiv quantifizierbarer Lernerfolg in Form einer erreichten Punktzahl und den hinzugewonnenen Stufenanstiegen gemessen und für den Nutzer dargestellt werden. Die Stufenanstiege sorgen zudem dafür, dass die Schwierigkeit der gestellten Fragen ansteigt [siehe 2.2.1](#).

Mit jeder Frage rückt der Lernende also dem Lernziel, alle vorhandenen Begriffe zu erlernen, sichtbar einen Schritt näher und wird stetig dazu angehalten, seine Leistung zu verbessern, um der steigenden Herausforderung gerecht zu werden und weiterhin Erfolgserlebnisse zu erhalten.

2.4.2 Kognitivismus

Die zweite wichtige Lerntheorie, die bei OWLisch Anwendung findet, ist der **Kognitivismus**. Dieses Konzept ging in den 1950er Jahren aus dem *Behaviorismus* hervor und beschäftigt sich intensiver mit der *Black Box*, die bis dahin größtenteils ignoriert wurde. Im Mittelpunkt steht dabei das Verständnis der kognitiven Prozesse im menschlichen Gehirn. [4](#)

4. Vgl. Höhne, Sebastian, 2017: Kognitivismus.
<http://www.lernpsychologie.net/lerntheorien/kognitivismus> (07.01.2017)

Aus den Erkenntnissen des Kognitivismus lassen sich für die Didaktik und Entwicklung von Lernsystemen folgende Regeln und Leitlinien ableiten, die auch (teilweise) bei OWLisch eingesetzt werden: 5

Aufmerksamkeit wecken:

Der Lernende muss seine Aufmerksamkeit auf den zu lernenden

- Gegenstand legen, damit überhaupt erst ein Lernen stattfinden kann. Dies kann z.B. durch besondere Hervorhebungen oder Abwechslungsreichtum erreicht werden.

Vorwissen aktivieren:

Für den Lernenden ist es von Vorteil, wenn neue Informationen stetig mit bereits im Langzeitgedächtnis vorhandenem Vorwissen verknüpft werden können.

Wahrnehmungsprozess unterstützen:

Eine besonders übersichtliche und einfache Darstellung der Lerninhalte begünstigt das Lernen, da Informationen schneller und effizienter aufgenommen werden können.

Speicherung im Gedächtnis verbessern:

Das Gelernte muss vom Kurz- ins Langzeitgedächtnis übergehen; Wiederholungen der Aufgaben oder anderweitige Anwendung neuer Informationen kann dabei helfen.

Wissen überprüfen und verbessern:

Durch Lernkontrollen muss es dem Lernenden ermöglicht werden, sein Wissen zu prüfen und vertiefen. Erfolge bei Lernkontrollen wirken zudem motivierend und verstärken das Lernverhalten positiv.

5. Vgl. Vontobel, Peter, 2006: *Didaktisches Design aus lernpsychologischer Sicht*, Seite 10 ff.
Pädagogische Hochschule Zürich.

Diese Prinzipien spielen aufgrund der geringen Komplexität der Lerninhalte von *OWLisch* zwar nur eine untergeordnete Rolle, lassen sich aber vor allem bei der visuellen Gestaltung vom Quiz und vom Wörterbuch gut integrieren.

So wird jede Frage im Quiz als individuelle Lerneinheit kompakt und übersichtlich auf einem einzigen Screen dargestellt, ohne dass der Nutzer auf dem Bildschirm scrollen oder anderweitig nach benötigten Informationen suchen muss. Eine Frage muss erst gelöst werden, bevor die nächste angezeigt wird und das Quiz kann zwischendurch nicht verlassen werden, ohne es vollständig abzubrechen. Dadurch findet keinerlei Ablenkung statt und die volle Aufmerksamkeit des Lernenden wird auf diese eine Aufgabe gelenkt, was sowohl *die Aufmerksamkeit weckt*, als auch *den Wahrnehmungsprozess unterstützt*.

Auch die unterschiedlichen Quiz-Typen [siehe 2.2.1](#) und die Implementierung der Audiofunktion zum Vorlesen der ostwestfälischen Begriffe kommen diesen beiden Prinzipien zugute, indem sie einerseit Abwechslung schaffen und so einer Reizmonotonie vorbeugen, und andererseits unterschiedliche Arten der Wahrnehmung und Verarbeitung von Informationen (Text, Bild und Ton) anbieten, die zudem die Verknüpfung der Lerninhalte mit bereits *vorhandenem Vorwissen* unterstützen (dem Lernenden mag ein Begriff noch nicht als geschriebenes Wort untergekommen sein, aber vielleicht hat er ihn schon einmal gehört oder kennt den abgebildeten Gegenstand). Das Prinzip *Vorwissen aktivieren* spielt im didaktischen Konzept der App allerdings nur eine unwesentliche Rolle, da die Lerninhalte eigentlich simpel genug sind, um keinerlei Vorwissen vorauszusetzen.

Das in *OWLisch* enthaltene persönliche Wörterbuch des Nutzers kann im kognitivistischen Sinne die *Speicherung im Gedächtnis verbessern*, da der Lernende hier jederzeit Zugriff auf die ihm bereits bekannten Begriffe hat und kleine Häppchen an zusätzlichen Informationen (Zitate, Anekdoten, Hintergrundwissen) zu diesen Begriffen abrufen kann, die als zusätzliche Erinnerungsstütze dienen. Auch beim Wörterbuch findet die oben genannte kompakte und ablenkungsfreie Gestaltung Anwendung.

Das Prinzip *Wissen überprüfen und verbessern* wird mit dem Quiz als Kernelement der App natürlich in vollem Umfang umgesetzt; es wird dem Nutzer nicht nur ermöglicht, eine Lernkontrolle durchzuführen, diese ist sogar der Hauptmechanismus des Lernprozesses. Hier überschneiden sich die Konzepte des *Behaviorismus* und die Erkenntnisse des *Kognitivismus*.

2.5 Ostwestfälische Begriffe

Zum Abschluss dieses Kapitels folgt noch eine Auflistung aller 30 in der finalen Version von *OWLisch* enthaltenen ostwestfälischen Begriffe mitsamt Übersetzung und Artikel:

- | | |
|--|---|
| 1. angeschickert – angetrunken | 16. nöhlen – meckern |
| 2. beömmeln – sich totlachen | 17. nönkern – Mittagsschlaf halten |
| 3. betuppen – betrügen | 18. öddelich – dreckig |
| 4. Bollerbuche, <i>die</i> – Jogginghose | 19. Pättkenschnüwer, <i>der</i> – Moped |
| 5. Bömsken, <i>das</i> – Bonbon | 20. pecken – kleben |
| 6. Bütterken, <i>das</i> – Butterbrot | 21. Pillepoppen, <i>die</i> – Kaulquappen |
| 7. dölmern – spielen | 22. Pinneken, <i>das</i> – Schnapsglas |
| 8. Dönekens, <i>die</i> – Anekdoten | 23. plästern – regnen |
| 9. fickerich – nervös | 24. Pläte, <i>die</i> – Glatze |
| 10. Knüpp, <i>der</i> – Knoten | 25. Plüdden, <i>die</i> – alte Klamotten |
| 11. Kropfzeug, <i>das</i> – Krempel | 26. Pölter, <i>der</i> – Schlafanzug |
| 12. Latüchte, <i>die</i> – Laterne | 27. ratzen – schlafen |
| 13. Mäse, <i>die</i> – Hintern | 28. Töffel, <i>der</i> – Trottel |
| 14. Möttke, <i>die</i> – Schlammb | 29. vermackeln – beschädigen |
| 15. Mürker, <i>der</i> – Maurer | 30. wullacken – schuften |

3 Gestaltung der App

Nachfolgend werden die Typografie, Farbgestaltung, das Layout und das Stylesheet der App erläutert.

3.1 Überblick & Screenshots

Bevor die Gestaltung der App im Detail präsentiert und erläutert wird, folgt zunächst ein grober Überblick, der mittels Screenshots aller im vorigen Kapitel erwähnten Features und Layout-Elemente vermittelt wird. Anschließend werden die Farbgestaltung, Typografie und schließlich alle Gestaltungselemente erläutert, bevor das Kapitel mit einer kurzen Erklärung der Umsetzung mittels Stylesheets und der dort verwendeten Techniken abschließt.

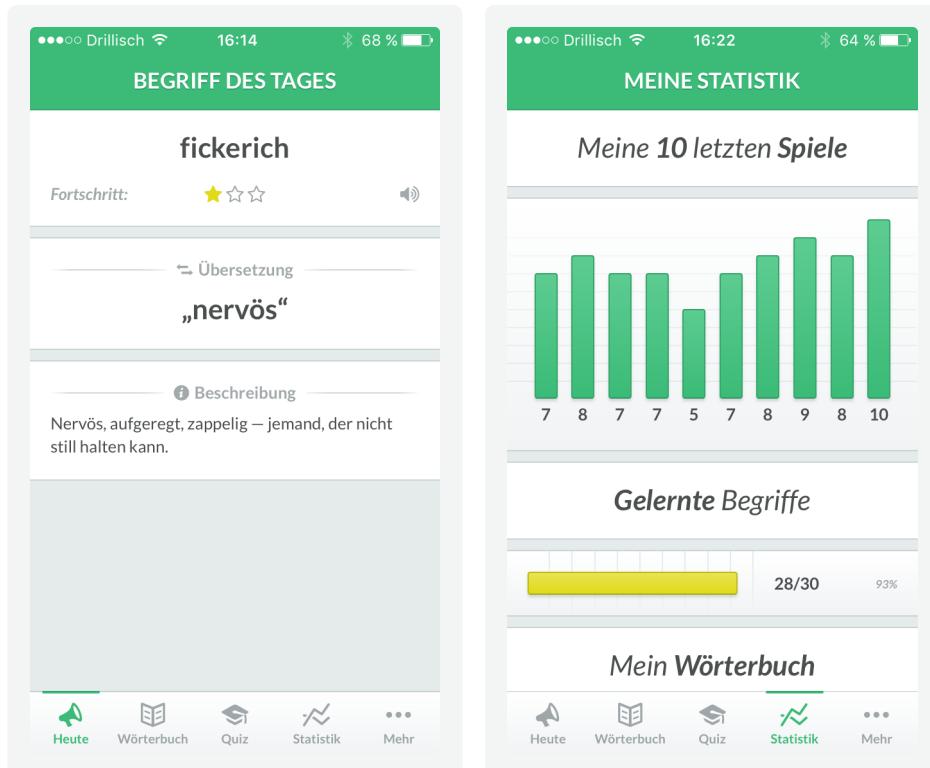


Abbildung 11

Screenshots: Begriff des Tages & Statistik

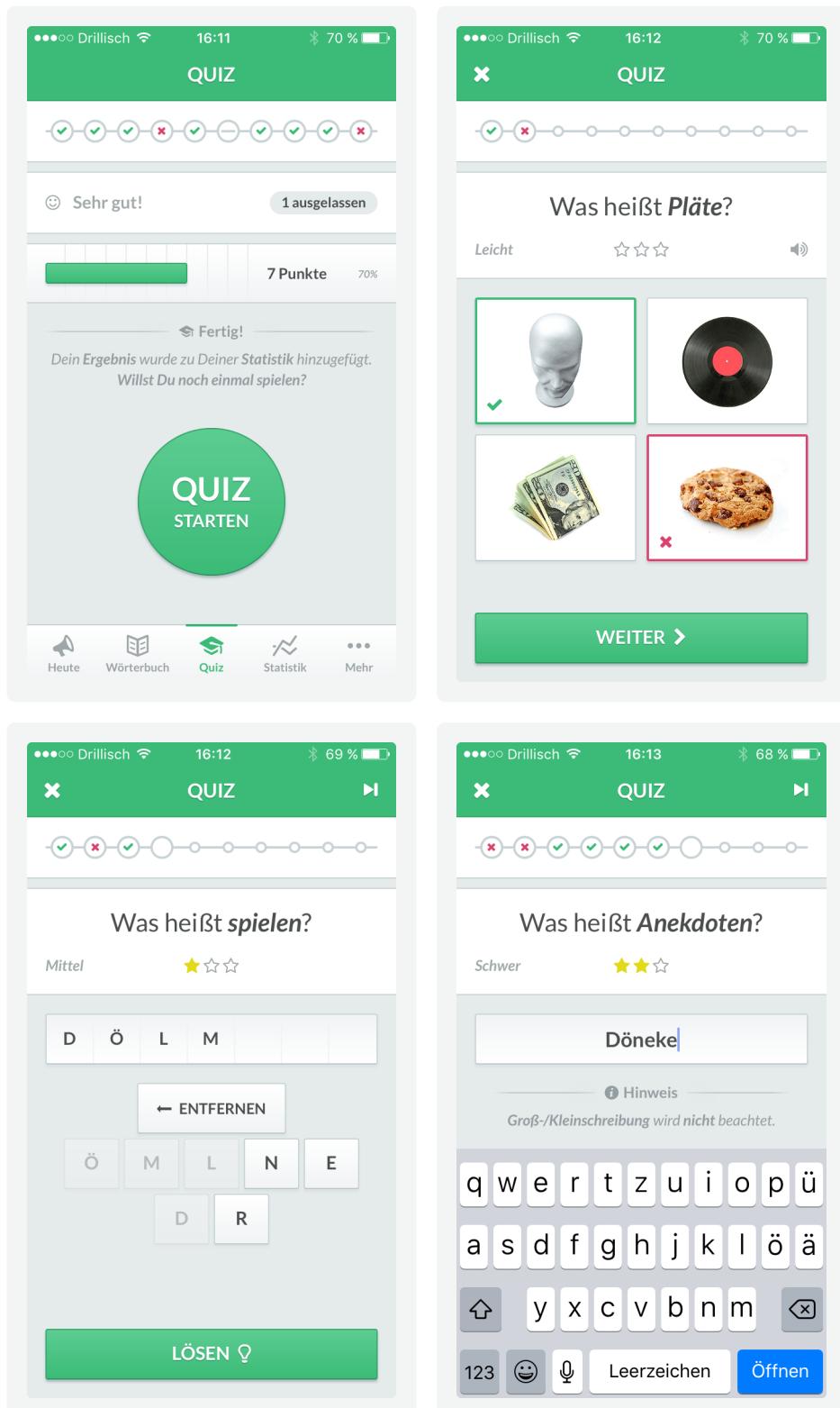


Abbildung 12

Screenshots: Quiz-Ende & -Typen

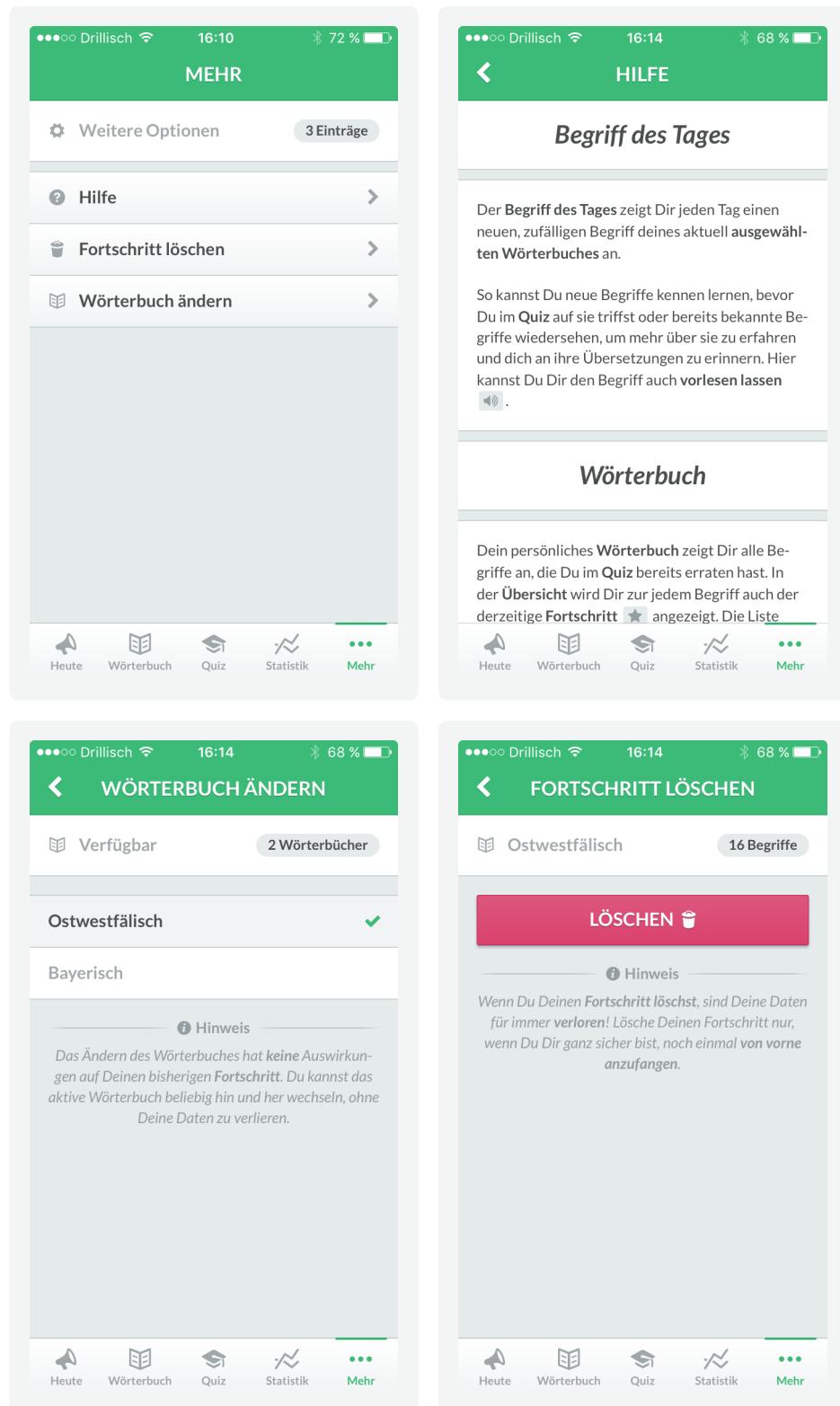


Abbildung 13

Screenshots: Mehr, Hilfe & Optionen

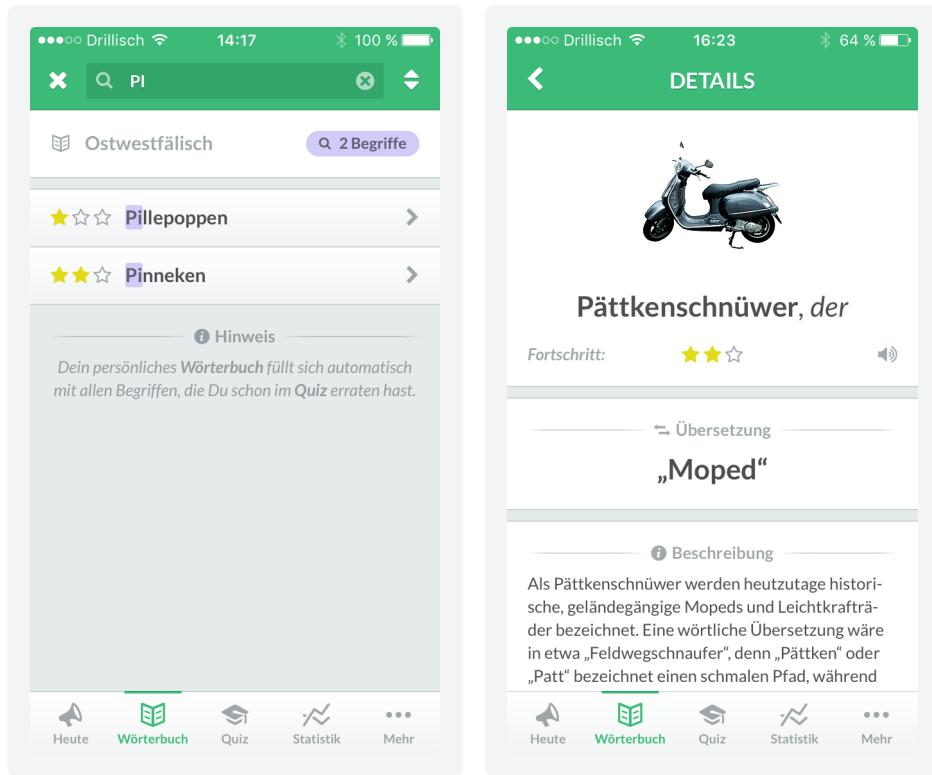


Abbildung 14

Screenshots: Wörterbuch-Übersicht & -Details

3.2 Farbgestaltung

Der Leitgedanke bei der Auswahl der Farbpalette für *OWLisch* war es, ein möglichst helles, freundliches und einladendes Interface für den Nutzer zu erstellen. Aus diesem Grund wurde, wie in den Screenshots [siehe Abbildung 11-14](#) zu sehen, hauptsächlich von den Farben Grün, Weiß und Grau Gebrauch gemacht, mit einem hellen Schwarz als Textfarbe, einem Rot als kontrastierende Warnfarbe (hauptsächlich im Quiz) und Gelb und einem hellen Violett als ergänzende Farben für Hervorhebungen von speziellen Funktionen. Dabei wird das Grün sowohl als Grundfarbe der App, als auch als Bestätigungsfarbe im Quiz verwendet und soll eine positive und entspannte Grundstimmung bei der Verwendung der App erzeugen, während das Rot als starker Kontrast zu den restlichen Farben eine besondere Signalwirkung entfaltet.

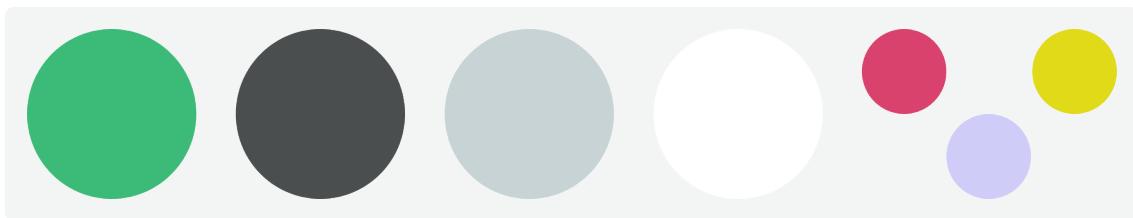


Abbildung 15

Farbpalette der App

Die Kombination aus *Grün* und *Rot* entspricht dem klassischen Schema von *Erfolg* und *Misserfolg*, durch den zusätzlichen Einsatz des gleichen *Grüns* als Hauptfarbe für die App wird der *Erfolg* jedoch zur Grundsituation bei der Benutzung von *OWLisch* erhoben, sodass der Kontrast des *Rot* stärker wahrgenommen wird und ein schnelles Erkennen von Fehlern im Quiz ermöglicht. Es wurde jedoch darauf geachtet, dass die beiden Farben in diesem Kontext stets von Symbolen begleitet sind, um die Bedeutung auch für Farbfehlsichtige vermitteln zu können [siehe Abbildung 12](#).

Die Farben *Gelb* und *Violett* dienen als visueller Hinweis für bestimmte Funktionalitäten innerhalb der App und werden nur in ihrem entsprechenden Kontext verwendet. Das *Gelb* steht dabei stets in Zusammenhang mit dem Fortschritt des Nutzers und taucht sowohl bei den Sternen, die die aktuelle Stufe eines Begriffes symbolisieren, als auch in den Diagrammen innerhalb der Statistik auf, die den Gesamtfortschritt des Lernenden darstellen [siehe Abbildung 11](#). Dabei wurde das *Gelb* aufgrund seiner Symbolik und Assoziation mit *Erleuchtung* und *Dynamik* gewählt.

Das *Violett* wird lediglich bei der Suchfunktion in der Wörterbuch-Übersicht verwendet [siehe Abbildung 14](#) und dient dort als Textmarkierung des Suchbegriffs innerhalb der Liste. Gleichzeitig stellt es eine visuelle Verbindung zu dem farblich identisch markierten Label mit der Anzahl der Suchbegriffe her und signalisiert dem Nutzer so, dass er sich gerade im *Suchmodus* befindet. Die Suchfunktion wird in Kapitel 3.4.2 näher erläutert.

3.3

Typografie & Icons

In der gesamten App (sowie in diesem Dokument) wird die Schriftart *Lato* (polnisch für *Sommer*) verwendet [siehe Abbildung 16](#); es handelt sich dabei um eine seriflose Open-Source Schriftart, die im Jahr 2010 vom Gestalter Łukasz Dziedzic in Warschau entwickelt und in Kooperation mit Google und der polnischen Schriftgießerei *tyPoland* veröffentlicht wurde.⁶

Lato Regular

**The quick brown fox jumps over the lazy dog.
The five boxing wizards jump quickly.**

Lato Bold

**The quick brown fox jumps over the lazy dog.
The five boxing wizards jump quickly.**

Lato Italic

**The quick brown fox jumps over the lazy dog.
The five boxing wizards jump quickly.**

Lato Bold Italic

**The quick brown fox jumps over the lazy dog.
The five boxing wizards jump quickly.**

Abbildung 16

Schriftart Lato

6. Vgl. Dziedzic, Łukasz, 2017: Lato.
<http://www.latofonts.com/lato-free-fonts> (13.01.2017)

Lato kommt dabei in den Schriftschnitten *Regular*, *Italic*, *Bold* und *Bold Italic* und den folgenden Schriftgrößen und deren Bezeichnungen im LESS-Quellcode (näheres dazu in Kapitel 3.5) innerhalb der App zum Einsatz:

- | | |
|------------------------------|-------------------------------|
| 1. 09px @app-font-size-micro | 4. 15px @app-font-size-medium |
| 2. 11px @app-font-size-tiny | 5. 18px @app-font-size-big |
| 3. 13px @app-font-size-small | 6. 22px @app-font-size-large |

Die Entscheidung, eine eigene Schriftart in *OWLisch* zu implementieren, entstand aus dem Vorhaben, die App auf mehreren Plattformen (*iOS*, *Android*, diverse Browser) zu unterstützen. Die Plattformen und unterschiedlichen Betriebssysteme unterstützen jedoch nicht die selben Standard-Schriftarten; um eine einheitliche Gestaltung zu erreichen, wurde daher *Lato* mit dem CSS Font-Face Feature implementiert.

Die Wahl von *Lato* stützt sich dabei auf zwei Haupt-Kriterien: Einerseits ist die Schriftart unter der *SIL Open Font License* veröffentlicht und ist daher kostenlos und frei in Projekten verwendbar, andererseits handelt es sich um einen modernen, unauffälligen und gut lesbaren Font, der für die Darstellung auf Bildschirmen (insbesondere für Apps und Websites) optimiert ist. Dezent Rundungen in der Schriftart geben ein Gefühl von Wärme, während die Geradlinigkeit und Struktur des Fonts Stabilität und Ernsthaftigkeit vermitteln, ohne dass diese Charakteristiken bewusst ins Auge fallen und den Nutzer vom eigentlichen Inhalt ablenken, was für die App, die in der Gestaltung ein besonderes Augenmerk auf Übersichtlichkeit und Simplizität legt, passende Merkmale für einen Font sind.⁷

7. Vgl. Dziedzic, Łukasz, 2017: *Lato*.
<http://www.latofonts.com/lato-free-fonts> (13.01.2017)

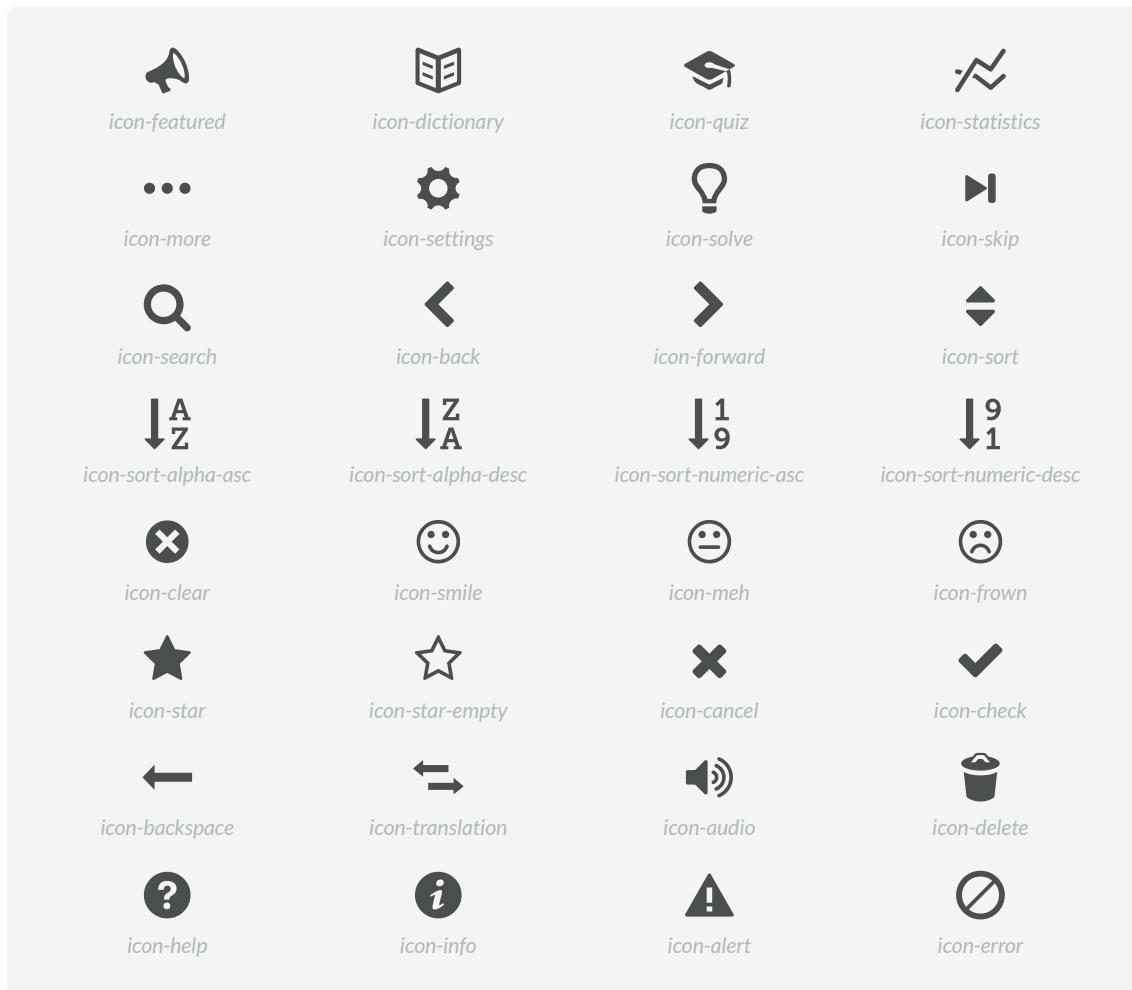


Abbildung 17

Übersicht der Icons in der App

Sämtliche in OWLisch enthaltenen Icons sind in obiger Abbildung dargestellt. Die Icons wurden jedoch nicht als Grafiken implementiert, sondern als Schriften, sogenannten *Webfont-Icons*. Dies hat gegenüber herkömmlichen Grafiken den Vorteil, dass die Symbole wesentlich flexibler eingesetzt und mittels CSS gestaltet werden können; Größe, Farbe und andere Eigenschaften lassen sich so analog zur herkömmlichen Typografie manipulieren, ohne für jeden Kontext eines Icons eine separate Grafik erstellen zu müssen.

Ein weiterer Vorteil ist die Unabhängigkeit von der Pixeldichte des Bildschirms, auf dem die *Webfont-Icons* dargestellt werden; herkömmliche Grafiken müssen so z.B. für *Retina Displays* oder andere Bildschirme mit hoher Pixeldichte zusätzlich in hochauflösenden Versionen bereitgestellt werden, um eine optimale Darstellung zu ermöglichen.

In der App kommen zwei unterschiedliche und miteinander zu einer Schrift-Datei zusammengefasste Icon-Pakete zum Einsatz: *Entypo* von *Daniel Bruce* und *Font Awesome* von *Dave Gandy*, beide unter der *SIL Open Font* Lizenz veröffentlicht und daher frei und kostenlos verwendbar. Die Icon-Pakete haben große Ähnlichkeit zueinander und lassen sich daher gut kombinieren; die Entscheidung, zwei Pakete zu benutzen, stammt aus der Tatsache, dass keines der beiden alleinstehend alle gewünschten Icons für *OWLisch* enthält.

Für das zusammengestellte Paket wurde jedes individuelle Symbol mit Hilfe des webbasierten Dienstes *Fontello*⁸ ausgewählt und zu einer einzelnen Datei zusammengefasst; es sind demnach keine Icons enthalten, die nicht auch in der App verwendet werden, was angesichts der Tatsache, dass diese Icon-Pakete in ihrer Gänze hunderte von Symbolen enthalten, eine merkliche Reduktion der Gesamt-Dateigröße von *OWLisch* zur Folge hat.

Stilistisch passen die Symbole sowohl zur Schriftart *Lato*, als auch zur restlichen Gestaltung von *OWLisch*; sie sind simpel und klar definiert, teilen sich die subtilen Rundungen mit *Lato* und vermitteln ihre Bedeutung auf den ersten Blick. Dabei dienen die Symbole einerseits dazu, dass Interface der App durch Details aufzulockern und interessanter zu gestalten, und andererseits zur Verdeutlichung und besseren Erkennbarkeit von Aktionen oder dargestellten Informationen. Häufig werden sie unterstützend neben Beschriftungen oder auf Schaltflächen platziert, wenn ihre Bedeutung ansonsten nicht unmittelbar erkennbar ist.

8. Siehe: <http://fontello.com> (13.01.2017)

3.4 Gestaltungselemente

Es folgt eine detaillierte Betrachtung individueller Gestaltungselemente innerhalb der App und eine Einordnung dieser Elemente in den Gesamtkontext ihrer Verwendung und Funktion.

3.4.1 Tab Bar

Die Gestaltung der *Tab Bar* entspricht im Wesentlichen dem bereits erwähnten Konzept [siehe 2.3.4](#), wurde jedoch um zwei Elemente erweitert: Ein zusätzlicher Indikator in Form eines grünen Balkens über dem aktiven Menüpunkt und unterstützende Beschriftungen der Symbole [siehe Abbildung 18](#). In frühen Tests hat sich herausgestellt, dass Icons alleine zu verwirrend für die Nutzer sind, da ihre Bedeutung nicht auf Anhieb erkennbar ist und der Nutzer so nicht weiß, was sich hinter welchem Menüpunkt verbirgt; zudem wird in den *iOS Human Interface Guidelines* eine Verwendung von Labels in der *Tab Bar* empfohlen. Der zusätzliche Indikator für den aktiven Menüpunkt dient hingegen schlicht der besseren Erkennbarkeit und Unterscheidung zu den restlichen Punkten.

Typisch für *iOS* sind hierbei inaktive Punkte in grau und der aktive Menüpunkt in der Hauptfarbe der App (in diesem Fall grün) dargestellt. Am oberen Rand der *Tab Bar* wurde eine dunkle Linie platziert, die zur besseren visuellen Trennung des Inhalts von der *Tab Bar* dient.

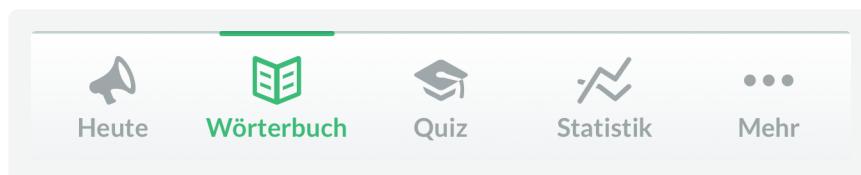


Abbildung 18

Detailansicht: Tab Bar

Die Anordnung der Menüpunkte innerhalb der *Tab Bar* folgt sowohl logischen, als auch ästhetischen Gesichtspunkten. Der erste Menüpunkt, der beim Starten der App angezeigt wird, ist der *Begriff des Tages*, der aufgrund des begrenzten Platzes in *Heute* umbenannt wurde und auch an erster Stelle, ganz links, platziert wurde. Der Punkt *Mehr*, hinter dem sich alle Menüpunkte mit niedriger Priorität verbergen, wurde hingegen an die letzte Stelle gesetzt, was somit der typischen Leserichtung von links nach rechts folgt.

Die verbleibenden drei Punkte wurden hingegen so angeordnet, dass sich aufgrund der Textlänge ihrer Beschriftungen eine Symmetrie um den Mittelpunkt der *Tab Bar* ergibt; *Wörterbuch* und *Statistik* haben mit zehn und neun Buchstaben eine ähnliche Länge und umschließen den zentralen Punkt *Quiz*, der nicht nur optisch, sondern auch logisch eine zentrale Position als Hauptfunktion der App einnimmt. Aus dieser mittleren Positionierung vom *Quiz* folgt auch eine Symmetrie mit anderen Elementen innerhalb der View, sobald der Nutzer das *Quiz* aufruft [siehe Abbildung 12](#).

3.4.2 Navigation Bar

Die Gestaltung der *Navigation Bar* entspricht ebenfalls zum Großteil dem zuvor beschriebenen Konzept [siehe 2.3.2](#), wurde jedoch um eine Suchfunktion für das *Wörterbuch* erweitert [siehe Abbildung 19](#). Die *Navigation Bar* wurde in der Hauptfarbe der App mit weißem Text darauf gestaltet und geht, typisch für iOS, nahtlos in die *Status Bar* über.

Die *Navigation Bar* enthält in *OWLisch* bis zu zwei Icons (links und rechts), die innerhalb des aktuellen Kontextes bestimmte Funktionen oder Aktionen erlauben, wie z.B. das Abbrechen oder Überspringen im *Quiz*, das Zurück-Navigieren innerhalb des *Wörterbuchs* oder das Durchsuchen oder Sortieren der *Wörterbuch-Übersicht*. Zudem befindet sich der Titel der aktuellen View mittig platziert in Großbuchstaben in dieser Leiste.

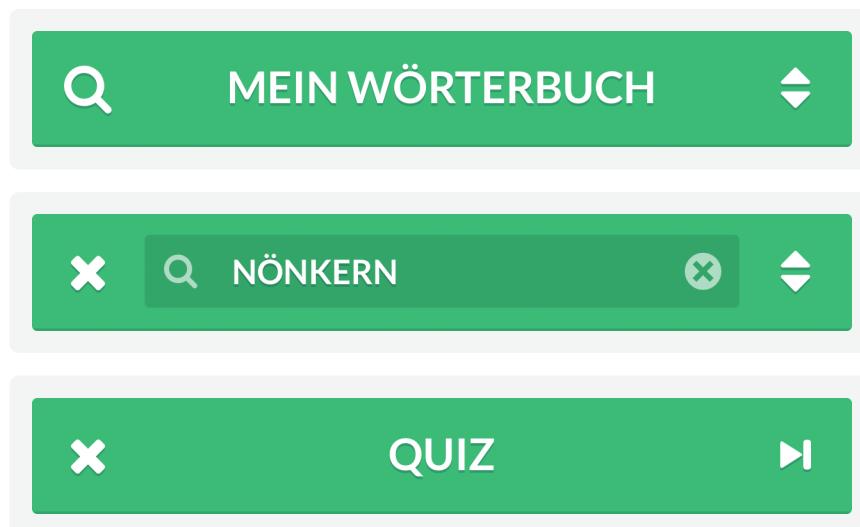


Abbildung 19

Detailansicht: Navigation Bar

Die Funktionalitäten im *Wörterbuch* sind dabei Elemente, die erst bei der eigentlichen Entwicklung der App im Anschluss an die Konzipierung im Detail ausgestaltet und implementiert wurden. Es wurde ersichtlich, dass eine Liste mit dutzenden oder gar hunderten von Begriffen schnell sehr unübersichtlich für den Nutzer werden kann, sodass ihm gleich zwei Möglichkeiten gegeben wurden, diese Liste effizienter zu navigieren: Einerseits kann mit dem Lupen-Symbol in der *Navigation Bar* ein Suchfeld eingeblendet werden, dass vorübergehend den Titel verdrängt und per Tastatureingabe eine Filterung der Liste in Echtzeit erlaubt, wobei nur noch Begriffe angezeigt werden, in denen der Suchbegriff enthalten ist [siehe Abbildung 14](#).

Zusätzlich dazu kann die Liste mit einem weiteren Button auf der rechten Seite der *Navigation Bar* sortiert werden; dabei wird eine Auswahlliste eingeblendet [siehe 3.4.5](#), die es dem Nutzer ermöglicht, das *Wörterbuch* sowohl alphabetisch, als auch nach Stufe des Begriffes zu sortieren, jeweils auf- und absteigend. Beide Möglichkeiten lassen sich dabei gleichzeitig einsetzen, sodass auch eine bereits gefilterte Liste zusätzlich nach Belieben sortiert werden kann.

3.4.3 Titel & Texte

OWLisch ist eine sehr textlastige App, *Titel* und *Texte* tauchen daher in diversen Variationen überall in der App auf; die gängigsten Varianten sind nebenstehend vergrößert dargestellt [siehe Abbildung 20](#). Dabei findet eine Unterscheidung zwischen *Titeln*, *Statustexten*, *Fließtexten* und *Hinweisen* statt:

Titel sind stets in schwarzem Text auf weißem Grund, mittig positioniert, mit einem großzügigen Innenabstand und in der größten verwendeten Schriftgröße dargestellt, um sie von restlichen Inhalten klar abzuheben. Ein *Titel* kann dabei von zusätzlichen Elementen begleitet werden, z.B. Informationen zum Fortschritt und der aktuellen Stufe eines Begriffes, einem Button zur Audio-Wiedergabe oder einem Bild zu einem Begriff. Teile des Textes innerhalb eines Titels können durch kursive oder fette Schriftschnitte besonders hervorgehoben werden.

Statustexte sind eine Form von Untertiteln, die zusätzliche hilfreiche Informationen im aktuellen Kontext darstellen, meistens am oberen Rand der View positioniert. Hier finden sich in der Regel numerische Angaben, z.B. zur Anzahl der vorhandenen Begriffe im *Wörterbuch*, die Anzahl der Fragen im *Quiz* oder das Ergebnis einer Quiz-Runde und die Anzahl der ausgelassenen Begriffe. Auf der linken Seite eines *Statustextes* befindet sich ein beschreibendes Label mitsamt Symbol in grauem Text, auf der rechten Seite die entsprechende numerische Angabe in einem grauen, abgerundeten Kasten. Eine weitere Art des *Statustextes* ist die Fortschritts-Leiste im *Quiz*; sie stellt in Form eines Diagramms die bisherigen Ergebnisse des Nutzers in der aktuellen Quiz-Runde dar und bietet somit zu jeder Zeit während eines Spiels eine Übersicht zum Fortschritt und den noch ausstehenden Fragen.

Die einzelnen Elemente innerhalb der Fortschrittsleiste können in fünf verschiedenen Varianten dargestellt werden, die sich alle die Charakteristik teilen, aus einem weißen Kreis mit grauen Rahmen zu bestehen, positioniert auf einer durchgehenden grauen Linie.

Dönekens, die

Fortschritt:  

Meine 10 letzten Spiele

i Beschreibung

Anekdoten, heitere Kurzgeschichten, oft von zweifelhaftem Wahrheitsgehalt.

i Hinweis

Du kannst das Quiz jederzeit abbrechen. Wenn Du eine Antwort nicht weißt, kannst Du die Frage auch auslassen. Je häufiger Du einen Begriff richtig errätst, desto schwieriger wird die Frage.

 Ostwestfälisch 5 Begriffe



Abbildung 20

Detailansicht: Titel & Texte

Noch ausstehende Fragen der Quiz-Runde werden dabei als kleiner Kreis angezeigt, während sowohl die aktuelle, als auch bereits beantwortete Fragen größer gestaltet sind. Die aktuelle Frage hat hierbei noch einen leeren, weißen Hintergrund, bei Beantwortung wird dieser Hintergrund mit einem entsprechenden Symbol gefüllt; ein grüner Haken für eine korrekte Antwort, ein rotes Kreuz für eine falsche, und eine graue Linie für eine übersprungene Frage.

Fließtexte und *Hinweise* sind sich strukturell ähnlich, werden aber unterschiedlich stark hervorgehoben. Gewöhnliche Texte werden auf weißem Grund, in schwarzer Schrift und linksbündig dargestellt, während *Hinweise* in grauer Schrift auf dem ebenfalls grauen Hintergrund der View, zentriert und kursiv gestaltet sind, um sie in den Hintergrund zu rücken und nicht unnötig Aufmerksamkeit auf sie zu lenken. *Hinweise* dienen als kleine zusätzliche Informations-Häppchen, die die Funktionsweise der App für neue Nutzer kurz und knapp am entsprechenden Ort erklären sollen, ohne zwingend die ausführlichere Hilfe der App aufrufen zu müssen. Erfahrene Nutzer können diese *Hinweise* aufgrund ihrer unauffälligen Gestaltung hingegen leicht ignorieren. Beide genannten Elemente können dabei zusätzlich um eine darüber befindliche Beschriftung mit Symbol erweitert werden, z.B. *Beschreibung* oder *Hinweis*, um auf ihre Funktion ersichtlich zu machen.

3.4.4 Buttons & Inputs

Buttons und *Inputs* sind in *OWLisch* vor allem im Quiz anzufinden und dienen dem Nutzer als Eingabe-Oberfläche zur Beantwortung von Fragen und Steuerung des Quiz-Verlaufs. Buttons sind dabei in verschiedenen Varianten mit unterschiedlichen Funktionen gestaltet: Antwort-Buttons verfügen stets über einen weißen Hintergrund (bei *Bilder*-Buttons) oder einem schwachen Verlauf von weiß nach hellgrau (bei *Text*-Buttons), mit einem grauen Rahmen und einem Schlagschatten, was ihnen als interaktives Element eine gewisse Räumlichkeit verleiht und somit stärker vom Hintergrund abhebt [siehe Abbildung 21](#).

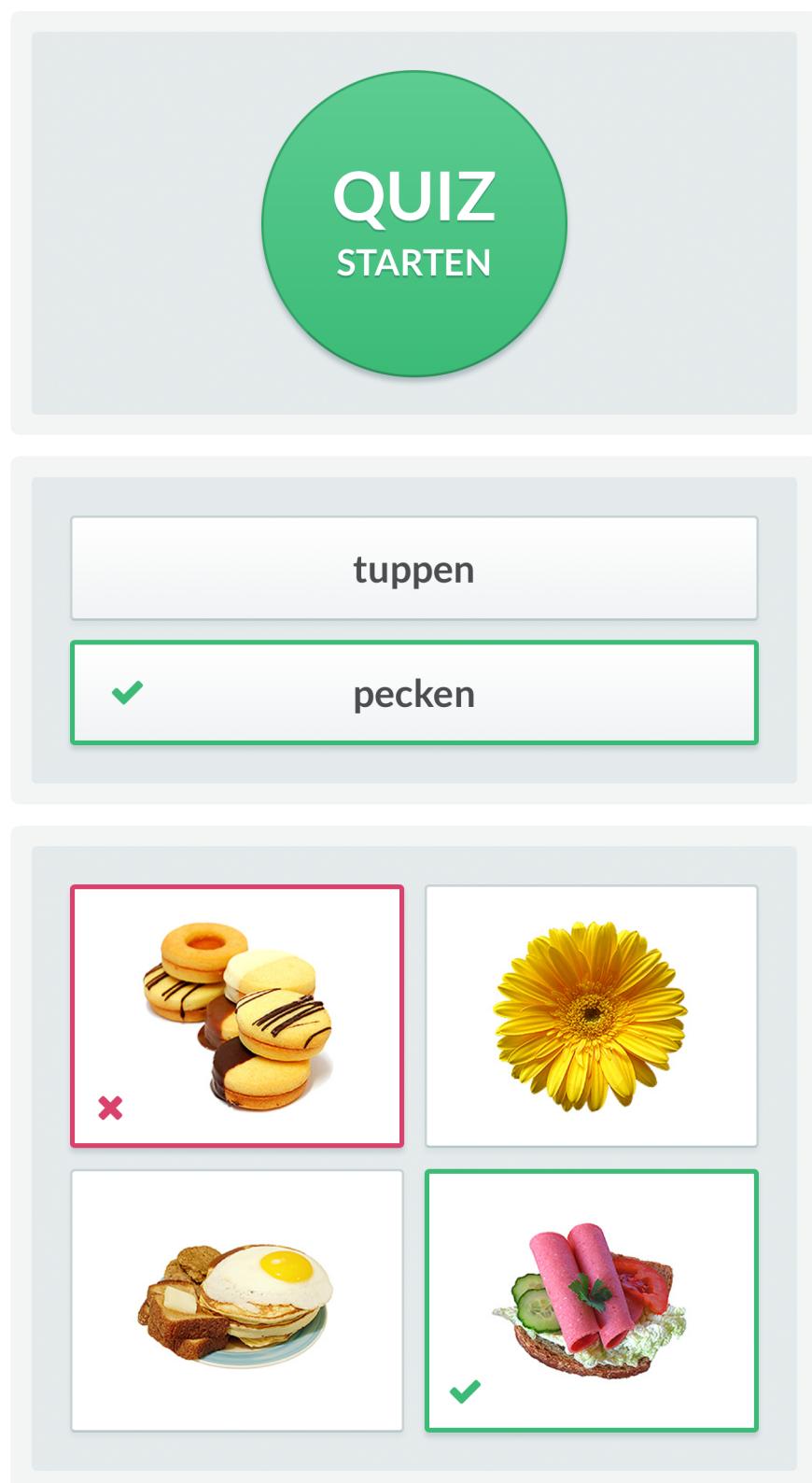


Abbildung 21

Detailansicht: Buttons

Diese Buttons können nach einer Interaktion seitens des Nutzers zudem verschiedene Zustände annehmen; sie zeigen durch die Ergänzung eines roten oder grünen Rahmens in Kombination mit einem ebenso gefärbten Symbol auf der linken Seite die Korrektheit der Antwort, indem die richtige Antwort grün und die falsche rot gefärbt wird. Dabei wird die falsche Antwort nur dann hervorgehoben, wenn sie vom Nutzer gegeben wurde, sodass zu sehen ist, welche Antwort ausgewählt wurde und welche richtig gewesen wäre.

Zusätzlich zu diesen Buttons gibt es noch *Aktions-Buttons*; diese verfügen stets über einen Hintergrund mit grünem Verlauf, weiße Schrift, einen dunklen Rahmen und einen Schlagschatten. Sie dienen dazu, das Quiz zu starten, zur nächsten Frage fortzufahren oder die aktuelle Frage zu lösen, falls es sich um eine Frage mit einer Texteingabe handelt. Durch die unterschiedliche Färbung dieser Buttons heben sie sich von den gewöhnlichen Antwort-Buttons ab und verdeutlichen, dass sie sich in ihrer Funktion unterscheiden. Eine Ausnahme unter den *Aktions-Buttons* stellt die Schaltfläche zum Starten des Quiz dar: Sie ist, im Gegensatz zu allen anderen Buttons, rund und besonders groß, was ihre spezielle Funktion zusätzlich betont und mehr Aufmerksamkeit darauf lenkt.

Inputs, also Eingabefelder, gleichen in ihrer Gestaltung den *Buttons*; während Schaltflächen jedoch nach *außen* gewölbt erscheinen, sind *Inputs* durch ihren Verlauf von hellgrau nach weiß nach *innen* gewölbt. *Inputs* tauchen in zwei Varianten auf: *Freitext-Felder*, in denen der Nutzer per eingeblendeter Tastatur beliebig Text eingeben kann, und indirekt gesteuerte und in einzelne Kästen aufgeteilte *Buchstaben-Felder*, die durch Interaktion mit damit verbundenen Buttons sequentiell befüllt oder entleert werden können [siehe Abbildung 22](#).

Auch diese Eingabefelder können, analog zu den Antwort-Buttons, nach einer erfolgten Interaktion des Nutzers verschiedene Zustände annehmen, die die Korrektheit der gegebenen Antwort mittels Farben und Symbolen signalisieren.

Hier findet der gleiche gefärbte Rahmen Anwendung, ergänzend kommt jedoch eine eingeblendete Meldung mit einem entsprechenden Symbol (grüner Haken oder rotes Kreuz) und der richtigen Antwort hinzu, um den Nutzer über seinen Erfolg oder Misserfolg aufzuklären. Vorhandene Buttons zum Befüllen des Inputs mit Buchstaben werden beim Einblenden der Meldung entfernt.

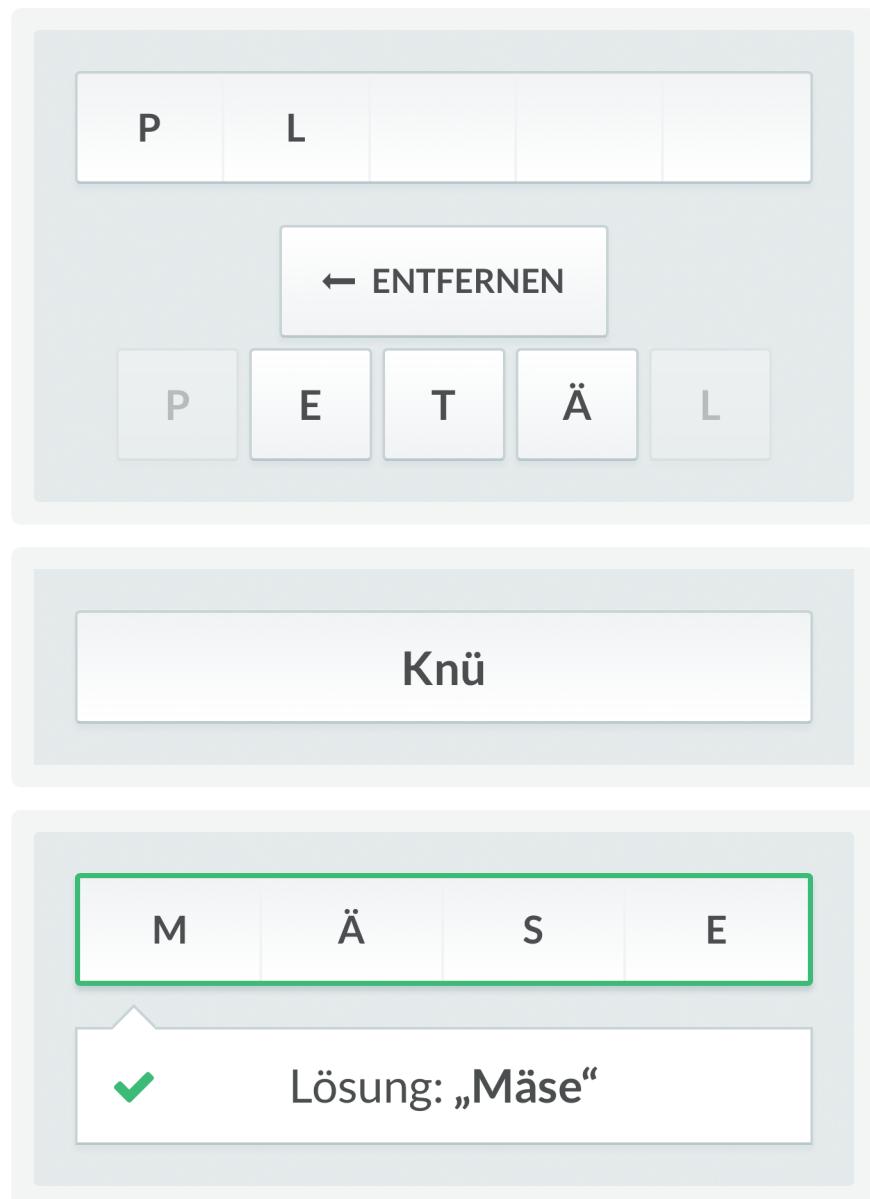


Abbildung 22

Detailansicht: Inputs

3.4.5 Listen

Listen treten in OWLisch in zwei Varianten in Erscheinung: *Navigations-Listen* und *Auswahl-Listen* [siehe Abbildung 23](#). Mit Hilfe von *Navigations-Listen* kann der Nutzer dabei innerhalb der aktuellen View in einen anderen Kontext navigieren, beispielsweise aus der *Wörterbuch-Übersicht* heraus zu den Details eines ausgewählten Begriffs, oder aus der Übersicht des Menüpunktes *Mehr* zu den dort vorhandenen zusätzlichen Optionen.

Jedes Listen-Element einer solchen *Liste* ist dabei identisch gestaltet; auf der linken Seite befindet sich (optional) ein Icon, das entweder zusätzliche Informationen bietet (wie die Stufe des Begriffes im Falle des *Wörterbuchs*) oder den Listen-Punkt repräsentiert. Daneben ist die eigentliche Bezeichnung des Elements platziert, während sich auf der rechten Seite ein Pfeil-Symbol befindet, das signalisiert, dass durch Interaktion mit dem Listen-Element innerhalb der App zu einem anderen Screen navigiert wird.

Die Listen-Elemente verfügen, ähnlich wie die zuvor beschriebenen *Buttons* [siehe 3.4.4](#), über einen nach außen gewölbten Verlauf, damit der Nutzer sie als Schaltflächen identifizieren kann. Zur besseren Orientierung bei der Interaktion mit der *Liste* wird der Wechsel der View animiert, indem der neue Inhalt von rechts nach links hereingefahren wird und den vorigen Inhalt überlagert. In Folge dessen wird auch die *Navigation Bar* angepasst, die nun auch über ein *Zurück*-Icon verfügt, über das der Nutzer zurück zur *Liste* gelangen kann.

Auswahl-Listen dienen in der App dazu, aus mehreren gegebenen Optionen eine davon als aktiv zu setzen; dies findet beim Sortieren der *Wörterbuch-Übersicht* [siehe 3.4.2](#) und beim Wechsel des *Wörterbuchs* Anwendung. Zur besseren Unterscheidung zu anderen Listen fehlt hier der erwähnte Verlauf. Inaktive Listen-Punkte verfügen über einen grauen Text, während der aktive Punkt einen grauen Hintergrund, schwarzen Text und einen grünen Haken hat, um ihn eindeutig identifizieren zu können.

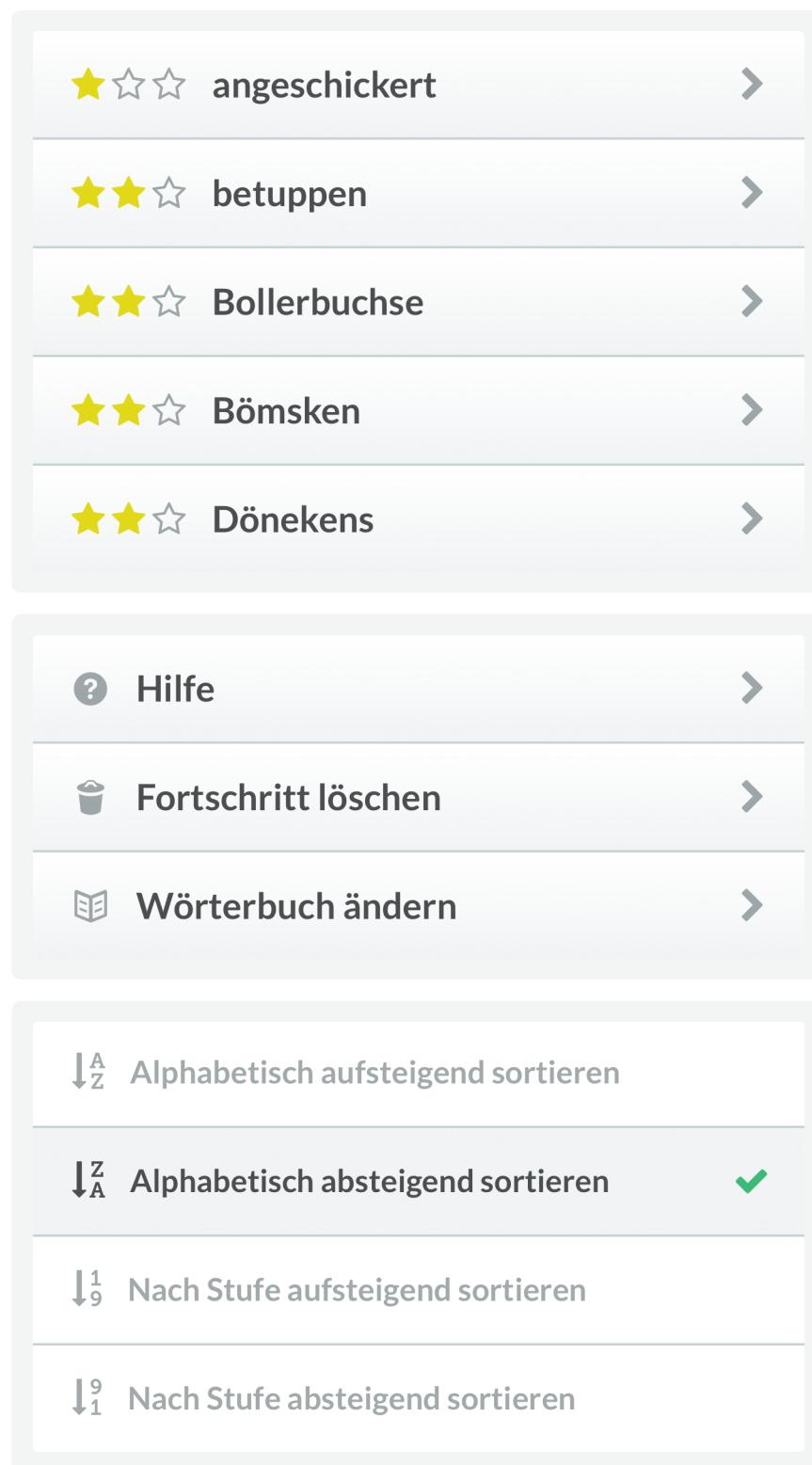


Abbildung 23

Detailansicht: Listen

3.4.6 **Diagramme**

Diagramme dienen in OWLisch als Visualisierung von numerischen Daten, in erster Linie in Verbindung mit den erzielten Ergebnissen im Quiz und dem erreichten Fortschritt des Nutzers in der Statistik. Zum Einsatz kommen dabei lediglich Balken-Diagramme, sowohl mit horizontalen, als auch vertikalen Balken, in zwei verschiedenen Farben [siehe Abbildung 24](#).

Die Balken selbst verfügen dabei über einen farbigen Hintergrund mit Verlauf, einen dunkleren Rahmen und einen Schlagschatten, um sie vom Hintergrund, der mit Hilfslinien versehen ist, plastisch abzuheben. Die unterschiedlichen Farben symbolisieren dabei verschiedene Kontexte: Grüne Diagramme stellen die erzielten Punkte in der aktuellen Quiz-Runde (am Ende jener Runde) oder die erzielten Punkte der letzten zehn Spiele dar. Da es sich beim erstgenannten Diagramm lediglich um einen Datenpunkt handelt, wurde hier ein horizontales Balken-Diagramm gewählt, während beim zweiten ein vertikales zum Einsatz kommt, um mehrere Datenpunkte nebeneinander darzustellen und optisch eine Kurve zu erzeugen, die die Leistung des Nutzers in den letzten Spielen visualisiert. Um die Punktzahlen auf den ersten Blick exakt bestimmen zu können, sind die einzelnen Balken zusätzlich mit eben jener Punktzahl beschriftet, im Falle des horizontalen *Diagramms* zusätzlich auch mit einer Prozentangabe.

Gelbe Diagramme zeigen den Gesamtfortschritt des Nutzers und teilen sich ihre Farbgestaltung somit mit den zuvor bereits erwähnten Sternen [siehe 3.2](#). Hier wurden horizontale Diagramme gewählt, die mit zusätzlichen Informationen beschriftet sind; die Anzahl der Stufen wird so z.B. noch einmal mit den Sternsymbolen selbst symbolisiert und die exakten und relativen Prozentzahlen finden sich, analog zu den grünen *Diagrammen*, ebenfalls als Achsenbeschriftungen im *Diagramm* wieder.

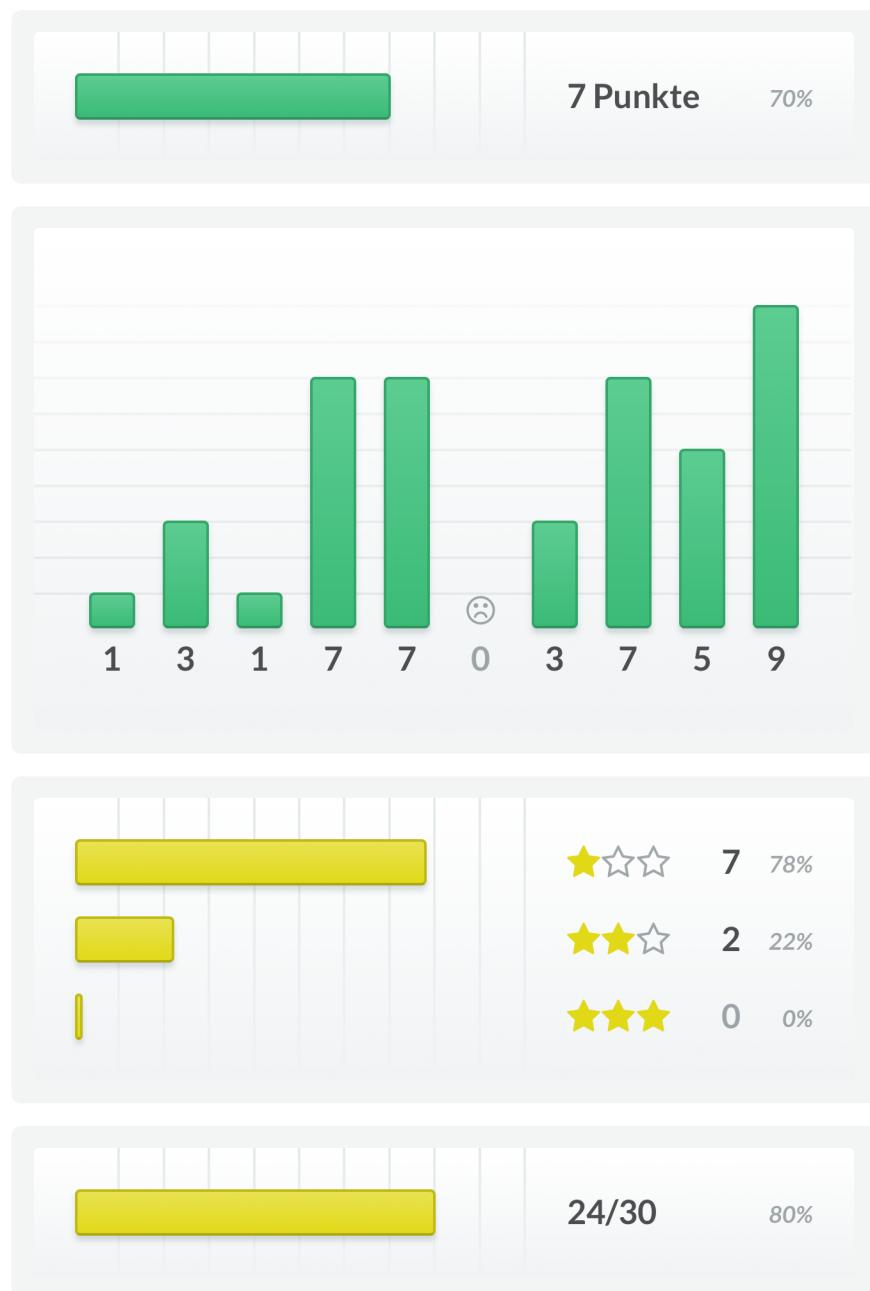


Abbildung 24

Detailansicht: Diagramme

3.4.7 App-Icon

Das App-Icon von OWLisch ist relativ simpel gestaltet; der Name der App in der Schriftart *Lato* mit weißem Text auf der Hauptfarbe der App, ergänzt durch einen flachen Schattenwurf [siehe Abbildung 25](#). Zur Verdeutlichung des Namens OWLisch wurde der Schriftzug in seine zwei Bestandteile *OWL* und *isch* aufgespalten; *OWL* wurde dabei oben in *Lato Black* platziert, während sich *isch* in *Lato Light Italic* darunter befindet und so visuell vom darüber befindlichen Schriftzug abgrenzt. Für die optimale Darstellung auf sämtlichen iOS-Geräten wurde das Icon in folgenden Größen zur Verfügung gestellt:

- | | | | |
|----|---------|-----|-----------|
| 1. | 29x29px | 9. | 87x87px |
| 2. | 40x40px | 10. | 100x100px |
| 3. | 50x50px | 11. | 114x114px |
| 4. | 57x57px | 12. | 120x120px |
| 5. | 58x58px | 13. | 144x144px |
| 6. | 72x72px | 14. | 152x152px |
| 7. | 76x76px | 15. | 167x167px |
| 8. | 80x80px | 16. | 180x180px |



Abbildung 25

App-Icon von OWLisch

Da OWLisch jedoch ebenfalls auf allen gängigen Web-Browsern funktionieren soll, wurde noch ein sogenanntes *Favicon* ergänzt; dieses wird von Browsern in der Adressleiste oder im entsprechenden Tab als Symbol für eine Website dargestellt und ist im Falle der App identisch mit dem eigentlichen App-Icon. Auch das *Favicon* muss in diversen Größen und Versionen zur Verfügung gestellt werden, hier wurde jedoch auf den webbasierten Dienst *Favicon Generator* ⁹ zurückgegriffen, der alle benötigten Varianten generiert.

9. Siehe: <http://realfavicongenerator.net> (15.01.2017)

3.5 Stylesheet

Zum Abschluss dieses Kapitels und als Übergang zum nächsten folgt eine kurze Erläuterung der eingesetzten Techniken und Technologien, mit denen die bisher in diesem Kapitel beschriebene und dargestellte Gestaltung von *OWLisch* letztlich in der App umgesetzt wurde.

Da die App, wie in der Einleitung erwähnt, mit Web-Technologien umgesetzt werden sollte, wurden sogenannte *Cascading Style Sheets*, kurz CSS, eingesetzt. CSS gilt als die Standard Stylesheet-Sprache im World Wide Web und erlaubt es, mittels einer Reihe von Eigenschaften und Anweisungen sämtliche Elemente einer Website (oder in diesem Fall einer App) zu gestalten.

Bei *OWLisch* kam zusätzlich zu gewöhnlichem CSS noch *LESS* CSS zum Einsatz; dabei handelt es sich um einen sogenannten *CSS-Präprozessor*, der CSS um eine Reihe von neuen Features erweitert und eine effizientere und flexiblere Entwicklung ermöglicht. Dabei fügt *LESS* der Sprache keine neuen Eigenschaften im eigentlichen Sinne hinzu, sondern ändert lediglich die Art, wie sie geschrieben wird, indem z.B. der Einsatz von Variablen und Verschachtelungen ermöglicht wird, um redundante Wiederholungen im Code zu minimieren. *LESS* wird durch einen *Compiler* stets zu regulärem CSS konvertiert.¹⁰

Ein Beispiel für den Einsatz von *LESS* sind die zuvor erwähnten Schriftgrößen innerhalb der App [siehe 3.3](#): Sie sind als Variablen mit eigenem Namen definiert und können so im gesamten Stylesheet des Projektes unter diesem Namen verwendet werden, ohne die entsprechenden Pixelwerte ständig zu wiederholen. So wird in *OWLisch* mit allen Werten verfahren; von Farben, über Schriftgrößen bis hin zu Innen- und Außenabständen, was die Umsetzung einer konsistenten Gestaltung erleichtert.

10. Vgl. Sellier, Alexis et al., 2017: *LESS: Getting started*.
<http://lesscss.org> (16.01.2017)

Um CSS in Kombination mit HTML einsetzen zu können, bedarf es sogenannter *Selektoren*, mit denen CSS-Eigenschaften auf ausgewählte HTML-Elemente angewandt werden können. In *OWLisch* wurden ausschließlich *Klassen-*Selektoren verwendet, z.B. `<nav class="tabbar">`, was mittels CSS mit `.tabbar { ... }` selektiert werden kann. Ein allgemeines Problem beim Entwickeln einer Website oder App ist dabei die Benennung dieser Klassen; hier kann sehr schnell Chaos und Unübersichtlichkeit entstehen, wenn keine konsistente Namenskonvention verwendet wird.

In *OWLisch* wurde daher auf die bekannte Namenskonvention *BEM*, kurz für *Block Element Modifier*, zurückgegriffen. Mit dieser Methode werden übergeordnete Elemente als *Blöcke* definiert, die in sich geschlossen und unabhängig von anderen *Blöcken* gestaltet werden und als eigenständige und wiederverwendbare Entitäten gelten. Innerhalb dieser *Blöcke* gibt es *Elemente*, die nicht eigenständig existieren können und stets zu ihrem übergeordneten *Block* gehören. Und schließlich gibt es *Modifikatoren*, die einen *Block* oder ein *Element* modifizieren, ohne eine neue Entität zu definieren:¹¹

`.block` `.block__element` `.block--modifier` `.block__element--modifier`

Ein Beispiel aus *OWLisch* für die Anwendung dieser Konvention ist die *Tab Bar*, die als *Block* gilt, während die darin enthaltenen *Tabs* *Elemente* darstellen und ein aktiver *Tab* um einen *Modifikator* erweitert wird:

```
<nav class="tabbar">
  <ul class="tabbar__list">
    <li class="tabbar__tab tabbar__tab--active">...</li>
    ...
  </ul>
</nav>
```

11. Vgl. Starkov, Vladimir; Strukchinsky, Vsevolod, 2017: *BEM – Block Element Modifier: Introduction*.
<http://getbem.com/introduction> (16.01.2017)

Für die Gestaltung dieses Beispiels mit CSS ergibt sich aus dem Zusammenspiel von *LESS* und *BEM* ein weiterer Vorteil; durch den Einsatz der erwähnten Verschachtelung und dem `&`-Operator in *LESS*, der sich stets auf den Selektor aus der übergeordneten Ebene bezieht, lassen sich unnötige Wiederholungen im Code reduzieren:

```
// CSS
.tabbar { ... }
.tabbar__tab { ... }
.tabbar__tab--active { ... }

// LESS
.tabbar {
  &__tab {
    &--active { ... }
  }
}
```

Zusätzlich zu dieser Namenskonvention wurden alle definierten *Blöcke* in separate Dateien aufgespalten, die mit *LESS* und der dazugehörigen Master-Datei `app.less` zu einer einzigen CSS-Datei kompiliert werden; dies erlaubt eine übersichtliche Strukturierung des Quellcodes und ein schnelles Auffinden von Code in den entsprechenden Dateien unter `src/less/app`:

- | | |
|-------------------------------------|--|
| 1. <code>app.less</code> | 8. <code>block-navigationbar.less</code> |
| 2. <code>block-badge.less</code> | 9. <code>block-notice.less</code> |
| 3. <code>block-body.less</code> | 10. <code>block-play.less</code> |
| 4. <code>block-button.less</code> | 11. <code>block-progressbar.less</code> |
| 5. <code>block-chart.less</code> | 12. <code>block-quiz.less</code> |
| 6. <code>block-dropdown.less</code> | 13. <code>block-slider.less</code> |
| 7. <code>block-list.less</code> | 14. <code>block-stars.less</code> |

- | | |
|--|--------------------------------------|
| 15. <code>block-statistics.less</code> | 18. <code>block-title.less</code> |
| 16. <code>block-tabbar.less</code> | 19. <code>block-view.less</code> |
| 17. <code>block-term.less</code> | 20. <code>block-viewport.less</code> |

Ergänzt werden diese Dateien durch mehrere Konfigurations-Dateien, in denen etwa Farben, Fonts, Variablen und Animationen definiert werden, die global gültig und damit unabhängig von den *Blöcken* sind:

- | | |
|--|---------------------------------------|
| 1. <code>config-animations.less</code> | 4. <code>config-mixins.less</code> |
| 2. <code>config-colors.less</code> | 5. <code>config-variables.less</code> |
| 3. <code>config-font.less</code> | |

Bei der Umsetzung der Gestaltung von *OWLisch* wurde außerdem darauf geachtet, ausschließlich CSS zu verwenden und nicht auf Grafiken zurückzugreifen; Schatten, Rundungen, Verläufe, Rahmen und Hintergründe werden mit CSS erzeugt und bleiben so skalierbar, unabhängig von der Größe oder Pixeldichte des Bildschirms und einfach zu manipulieren, während gleichzeitig die Dateigröße der App drastisch reduziert wird. Tatsächliche Grafiken, wie etwa die Bilder im Quiz [siehe Abbildung 21](#), liegen natürlich dennoch als `.jpg`-Dateien vor.

Für die Kompilierung und Minimierung von *LESS*-Code und das Zusammenfügen der Dateien wurde *Gulp*, ein sogenannter *Task-Runner*, verwendet; die genannten Aufgaben werden so voll automatisiert bei jeder Änderung einer Quell-Datei ausgeführt. Da *Gulp* auch für den restlichen Code der App von Bedeutung ist, folgt eine nähere Erklärung erst im nächsten Kapitel dieser Arbeit. Der vollständige *LESS*-Quellcode der App findet sich im Anhang.

4 Programmierung der App

Nachfolgend wird die Programmierung der App mit JavaScript, HTML und Cordova grob angerissen.

4.1 Fremdcode & Frameworks

Bevor die eigentliche Programmierung von OWLisch näher erläutert wird, folgt zunächst ein Überblick über den Fremdcode und die Frameworks, die bei der Entwicklung verwendet wurden. Im JavaScript-Code innerhalb der App selbst wurde dabei auf vier Plugins zurückgegriffen, die alle auf GitHub¹² zur Verfügung stehen und unter der MIT Lizenz veröffentlicht wurden:

`jQuery` von *The jQuery Foundation*:

Hierbei handelt es sich um eine umfangreiche Bibliothek an Hilfsfunktionen, die das Manipulieren und Verändern des HTML-Dokuments erleichtern und beschleunigen; eine der weltweit am häufigsten genutzten JavaScript-Bibliotheken.

`FastClick` von *FT Labs*:

Für gewöhnlich gibt es bei Browsern auf Mobilgeräten eine Verzögerung von etwa 300 ms zwischen dem Berühren des Bildschirms und dem Ausführen der damit verbundenen Aktion, um zu erkennen, ob der Nutzer einen Doppelklick ausführt. `FastClick` entfernt diese Verzögerung, die es bei nativen Apps nicht gibt.

`jquery-bemhelpers` von *Max Shirshin*:

Ein kleines Plugin für `jQuery`, das den Umgang mit der CSS-Namenskonvention *BEM* siehe 3.5 in JavaScript erleichtert.

12. Siehe: <https://github.com> (18.01.2017)

`mustache.js` von Jan Lehnardt:

Eine sogenannte *Templating-Engine*, die es ermöglicht, mittels Vorlagen dynamisch HTML-Inhalt in die App einzufügen; der Name entstammt der Ähnlichkeit der angewandten Syntax `{}{...}{}{}` mit einem Schnurrbart.

Was *OWLisch*, das bis zu diesem Punkt lediglich eine gewöhnliche Website oder Webapp ist, aber zu einer echten App macht, die auf iOS-Geräten installiert und theoretisch über den *Apple App-Store* heruntergeladen werden kann, ist das Framework *Cordova* von *The Apache Software Foundation*. Dabei handelt es sich um ein frei verfügbares Open-Source Framework, veröffentlicht unter der *Apache-2.0* Lizenz, das eine mit JavaScript, HTML und CSS programmierte Webapp als native App verpackt und dabei alle verbeiteten Plattformen und mobilen Betriebssysteme wie *iOS*, *Android* und *Windows Phone* unterstützt. ¹³

Die Installation und Ausführung von *Cordova* erfolgt dabei über die JavaScript-Laufzeitumgebung *node.js* ¹⁴ und dem darin enthaltenen Paketverwaltungstool *NPM (Node Package Manager)*. Im Falle von *OWLisch* wird durch dieses Framework ein Projekt für die Entwicklungsumgebung *Xcode* von *Apple* erzeugt, über die, in Kombination mit einem gültigen *Apple-Entwickler-Account*, die Installation auf einem *iPhone* oder das Ausführen in einem *iOS-Emulator* stattfinden kann (was jedoch nur auf dem *macOS* Betriebssystem funktioniert). Neben *Cordova* selbst wurden noch offizielle Plugins des selben Entwicklers für das Framework verwendet, die zusätzliche Funktionen innerhalb der App aktivieren:

1. `cordova-plugin-dialog`
2. `cordova-plugin-keyboard`
3. `cordova-plugin-statusbar`

13. Vgl. *The Apache Software Foundation*, 2017: *Apache Cordova*.

<https://cordova.apache.org> (18.01.2017)

14. Siehe: <https://nodejs.org> (18.01.2017)

4.2 Module & Struktur

Bei der Programmierung von OWLisch wurden die verschiedenen Funktionalitäten der App in *Module* gegliedert, die größtenteils in sich geschlossen sind und unabhängig von anderen *Modulen* agieren. Dabei gibt es *Kernmodule*, die spezifische Elemente der App, wie z.B. das *Quiz* oder die *Tab Bar*, eigenständig steuern, und *Hilfsmodule*, die zusätzliche und global verfügbare Funktionalitäten für alle *Module* bereitstellen. Dabei wurde jedes *Modul* in eine eigene Datei ausgelagert, wobei sämtlicher Quellcode mittels *Gulp* schließlich zu einzelnen Datei zusammengefügt wird [siehe 4.4](#). Die *Kernmodule* umfassen folgende Dateien und deren Funktionen:

1. `app.js` Initialisiert und startet die App
2. `data.js` Lädt und speichert lokale Daten und Dateien
3. `dictionary.js` Steuert das *Wörterbuch* der App
4. `featured.js` Steuert den *Begriff des Tages*
5. `more.js` Steuert den *Mehr*-Tab der App
6. `navigationbar.js` Steuert die *Navigation Bar* der App
7. `quiz.js` Steuert das *Quiz* der App
8. `statistics.js` Steuert die *Statistik* der App
9. `tabbar.js` Steuert die *Tab Bar* der App
10. `view.js` Steuert die *View* der App

Diese *Module* wurden im sogenannten *Revealing Module Pattern* programmiert; die Variablen, Methoden und Funktionen der *Module* sind dadurch vor Zugriff von außen geschützt, es sei denn, sie werden über eine öffentliche Schnittstelle offenbart. Eine Implementierung dieser Methode sieht beispielhaft folgendermaßen aus:

```
var Modul = (function() {
    function _private() { ... }      // Private Funktion
    function public() { ... }       // Öffentliche Funktion
    return { public: public };     // Öffentliche Schnittstelle
})();
```

Die individuellen *Kernmodule* können dabei nicht direkt miteinander interagieren oder kommunizieren, sondern sind voneinander abgeschottet und funktionieren auch ohne die Existenz anderer *Kernmodule*. Für den Austausch von Daten und Nachrichten zwischen ihnen steht stattdessen ein sogenannter *Mediator* zur Verfügung, der in *OWLisch* zu den *Hilfsmodulen* gehört; jedes *Modul* kann dabei beim *Mediator* über *Kanäle* Nachrichten senden und wiederum selbst *Kanäle* abonnieren, ohne zu wissen, welche *Module* auf Nachrichten reagieren oder überhaupt existieren.

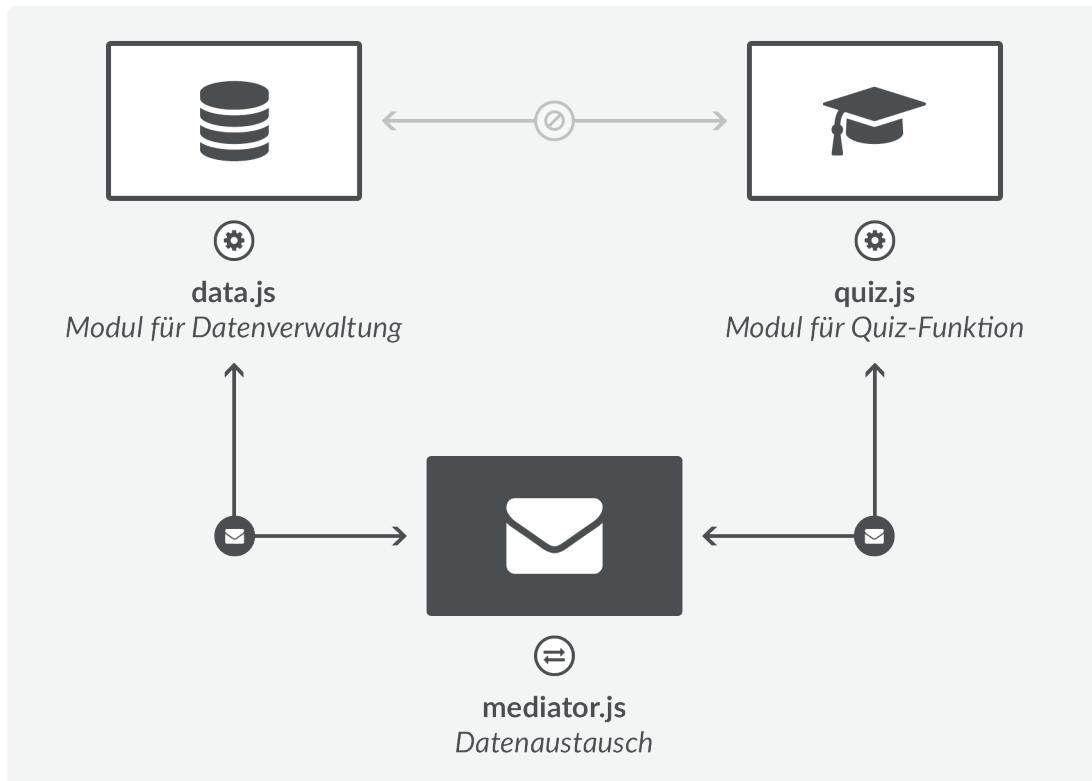


Abbildung 26

Flussdiagramm zum Mediator-Pattern

Im hier gezeigten Beispiel [siehe Abbildung 26](#) wird schematisch die Kommunikation zwischen dem *Data-Modul* und dem *Quiz-Modul* verdeutlicht; wird beispielsweise vom Nutzer während einer Quiz-Runde ein Begriff korrekt erraten, wird vom *Quiz-Modul* eine Nachricht mit einem Datenanhang im Kanal `terms:update` über den *Mediator* gesendet. Das *Data-Modul* reagiert auf diese Nachricht, indem die neuen Daten gespeichert werden und wiederum eine *Mediator*-Nachricht mit den aktualisierten Daten gesendet wird, die von mehreren anderen *Modulen* genutzt wird, um ihre eigene Anzeige zu aktualisieren (das *Statistics-Modul* aktualisiert so z.B. die dargestellten Diagramme).

Auf diese Weise bleibt die App flexibel und skalierbar; es können beliebige *Module* entfernt oder hinzugefügt werden, ohne die vorhandenen umschreiben zu müssen. Gesendete Nachrichten über den *Mediator* können von beliebig vielen *Modulen* empfangen und eigenständig verarbeitet werden, ohne mit anderen Elementen der App zu kollidieren, und neue *Module* müssen lediglich mit dem *Mediator* verknüpft werden, um die App um neue Funktionalitäten zu erweitern.

Neben den bereits genannten *Kernmodulen* gibt es in *OWLisch* noch folgende *Hilfsmodule*, die zusätzliche, allgemeingültige Funktionen bereitstellen (wie der *Mediator*). Für eine ausführliche Beschreibung aller hier genannten *Module* und den darin enthaltenen Methoden und Funktionen steht im Anhang dieser Arbeit eine umfangreiche JavaScript-Dokumentation zur Verfügung.

1. `config.js` Globale Konfigurationen und Einstellungen
2. `mediator.js` Steuert Kommunikation zwischen Modulen
3. `play.js` Steuert das Abspielen von Audio-Dateien
4. `slider.js` Stellt Funktionen für animierte Slider bereit
5. `template.js` Fügt Inhalte mit *mustache.js* in die App ein
6. `util.js` Allgemeine Hilfsfunktionen

4.3 Daten & Dateien

Neben den bisher beschriebenen JavaScript-Dateien, die den Kern des Programms bilden, und den in vorigen Kapiteln beschriebenen Favicons/App-Icons und CSS-Dateien werden sowohl beim Start, als auch während der Ausführung der App zusätzliche Dateien geladen und Daten geschrieben.

Einerseits handelt es sich dabei um *Template*-Dateien; dies sind einzelne HTML-Dateien, die bestimmte Elemente der App und zusätzliche Anweisungen für *Mustache* beinhalten, um dynamisch Inhalte einzufügen und die Struktur der App zum Start zu erzeugen. Diese Templates werden zu Beginn per AJAX (*Asynchronous JavaScript and XML*) geladen und vom *Template-Modul* zwischengespeichert. Ein Beispiel für ein solches Template ist `tmpl/tabbar.html`; diesem wird innerhalb des *TabBar-Moduls* eine Liste mit den in der App vorhandenen Tabs übergeben, woraufhin die fertig generierte *Tab Bar* in die App eingefügt wird:

```
<ul class="tabbar__list tabbar__list--hidden" data-tabbar="list">
  {{#.}}
    <li class="tabbar__tab" data-tabbar="tab" data-panel="{{name}}>
      <i class="tabbar__icon icon icon--{{icon}}></i>
      <span class="tabbar__label">{{label}}</span>
    </li>
  {{/.}}
</ul>
<div class="tabbar__status"></div>
```

Um die Daten der Wörterbücher, die in *OWLisch* zur Verfügung stehen, von der eigentlichen Programmierung der App zu trennen, wurden zusätzliche Dateien im `JSON` -Format (*JavaScript Object Notation*) angelegt, die ebenfalls per AJAX zu Beginn geladen werden. Dabei verfügt jedes Wörterbuch (z.B. *Ostwestfälisch*) über ein eigenes Verzeichnis, in dem sowohl die `JSON` -Datei, als auch die mit dem Wörterbuch verknüpften Bild- und Audio-Dateien enthalten sind.

Die Bild-Dateien liegen als `.jpeg` mit einer Größe von `280x280px` und die Audio-Dateien im `.mp3`-Format vor. Sämtliche in der finalen Version der App enthaltenen Bilder stammen aus der freien Bild-Datenbank *Freelimages*¹⁵, alle Audio-Dateien wurden freundlicherweise von *Miriam Belke*, die schon an der Entwicklung des ersten Prototypen beteiligt war [siehe 2.1](#), eingesprochen und bearbeitet. Die `JSON`-Datei eines Wörterbuches enthält alle für das Wörterbuch definierten Begriffe mitsamt Übersetzung, Beschreibungstext und Antwortmöglichkeiten im Quiz. Die enthaltenen Begriffe müssen dabei einem bestimmten Format folgen:

```
{
  "alias"          : "kurzname",
  "article"        : "Artikel, optional",
  "term"           : "Anzeigename",
  "translation"    : "Übersetzung",
  "info"           : "Beschreibungstext",
  "answersNative"  : ["Deutsche Antwort", "..."],
  "answersForeign" : ["Fremdsprachen-Antwort", "..."],
  "answersPictures": ["Bild-Datei", ...]
}
```

Die Datei selbst (z.B. `data/owl/owl.json`) kann beliebig viele Begriffe enthalten und die App kann beliebig viele Wörterbücher enthalten, wodurch *OWLisch* sehr leicht erweiterbar ist. Testweise ist in der App auch ein bayerisches Wörterbuch mit zehn Begriffen enthalten, das über den *Mehr*-Tab ausgewählt werden kann [siehe Abbildung 13](#). Die `JSON`-Dateien und Bilder werden mit *Gulp* zusätzlich minimiert/komprimiert und bei der Ausführung von *OWLisch* im Browser nur bei Bedarf geladen, um die Datenlast und Ladezeit für den Nutzer zu reduzieren. In der Anwendung als installierte App liegen alle Dateien ohnehin lokal auf dem Gerät vor, sodass keine merklichen Ladezeiten entstehen.

15. Siehe: <http://www.freeimages.com> (18.01.2017)

Die letzte Art von Daten, die in *OWLisch* vorkommen, sind die Fortschritts-Daten des Nutzers; diese werden jedoch nicht als Dateien gespeichert, sondern im sogenannten *LocalStorage* des Browsers. Dieser verhält sich ähnlich zum *Cache*, nur dass hier auch persistente Daten (in einem begrenzten Rahmen) abgelegt werden können, die bei jedem erneuten Besuch der App (in der Ausführung als Browser-App) weiterhin vorhanden sind.

Bei der Konvertierung von *OWLisch* zu einer nativen App mittels *Cordova* dient der *LocalStorage* jedoch ebenfalls als geeigneter Speicherort für die Daten; sie werden dann lokal auf dem Gerät des Benutzers gespeichert und können in *iOS* sogar per *iCloud* zwischen mehreren Geräten synchronisiert werden. Zwar liegt das Maximum der gespeicherten Daten bei etwa 5MB, was in Anbetracht der geringen Datenmengen von *OWLisch* jedoch kein Problem darstellt.

Gespeichert werden auf diese Weise die Ergebnisse der zehn letzten Quiz-Runden des Nutzers, die aktuelle Stufe jedes Begriffes aus dem Wörterbuch, der Begriff des Tages mit Datum (jeweils separat für jedes vorhandene Wörterbuch) und das aktuell verwendete Wörterbuch. Beispiellohaft können diese Daten folgendermaßen aussehen:

```
{  
  "dictionary": "owl",  
  "owl": {  
    "featured": { "term": "maese", "date": 20170118 },  
    "progress": { "maese": { "lvl": 3, "fail": 1 }, ... },  
    "scores" : [1, 3, 1, 7, 7, 0, 3, 7, 5, 9]  
  }  
}
```

Innerhalb der App können diese Daten durch den Benutzer auch jederzeit gelöscht werden, um noch einmal von vorne anzufangen [siehe Abbildung 13](#). Beim Ändern des Wörterbuches bleiben die Daten jedoch verlustfrei bestehen, sodass der Nutzer beliebig hin und her wechseln kann.

4.4 Sonstige Hilfsmittel

Zum Abschluss dieses Kapitels folgt noch eine Aufzählung aller sonstigen Hilfsmittel, die bei der Entwicklung der App, der zugehörigen Dokumentation oder beim Erstellen dieser Arbeit zwar von Bedeutung waren, jedoch nichts mit der eigentlichen Funktion von *OWLisch* zu tun haben.

Als erstes sei hier das bereits erwähnte *Gulp* zu nennen; es handelt sich dabei um einen populären *Task-Runner*, der mittels *node.js* installiert und betrieben wird. *Gulp* ermöglicht es, mit einer selbst erstellen JavaScript-Datei und zusätzlichen Plugins Arbeitsabläufe zu definieren, die bei jeder Änderung einer Quell-Datei automatisiert ausgeführt werden und so etwa *LESS*-Dateien zu CSS kompilieren, JavaScript-Dateien zusammenfügen und minimieren, Bilder komprimieren oder einfach Dateien an einen anderen Ort kopieren.

Ebenfalls zum Einsatz kam die Versionierungs-Software *Git*; dabei handelt es sich um ein Werkzeug, mit dem Änderungen am Projekt dokumentiert und gesichert werden können, sodass zu jedem Zeitpunkt zu einer vorhergegangenen Version zurückgekehrt werden kann.

Für die Erstellung der bereits erwähnten im Anhang befindlichen JavaScript-Dokumentation wurde das Kommandozeilen-Programm *jsdoc-to-markdown* verwendet, das aus den Code-Kommentaren aller Quell-Dateien automatisiert ein detailliertes Dokument generiert.

An dieser Stelle sei auch erwähnt, dass das vorliegende Dokument mit HTML, CSS und dem Kommandozeilen-Programm *Prince*¹⁶ von YesLogic Pty. Ltd. erstellt und gestaltet wurde. Auch bei diesem Prozess wurde *Gulp* genutzt, um die Erstellung eines HTML-Dokuments und dessen Konvertierung zu einer PDF-Datei mit *Prince* zu automatisieren.

16. Siehe: <https://www.princexml.com> (18.01.2017)

5 Fazit

Als abschließendes Kapitel folgt ein Fazit und eine Zusammenfassung der vorliegenden Arbeit.

A

Literaturverzeichnis

1. Apple Inc., 2016: iOS Human Interface Guidelines.
<https://developer.apple.com/ios/human-interface-guidelines> (15.12.2016)
2. Dziedzic, Łukasz, 2017: Lato.
<http://www.latofonts.com/lato-free-fonts> (13.01.2017)
3. Höhne, Sebastian, 2017: Kognitivismus.
<http://www.lernpsychologie.net/lerntheorien/kognitivismus> (07.01.2017)
4. Lippische Landes-Zeitung, 2013: Die Unvergänglichkeit des Ostwestfälischen.
http://www.lz.de/ueberregional/owl/7860369_Die-Unvergaenglichkeit-des-Ostwestfaelischen.html (12.12.2016)
5. Meir, Susanne, 2006: E-Learning Plus.
https://lehrerfortbildung-bw.de/moodle-info/schule/einfuehrung/material/2_meir_9-19.pdf (07.01.2017)
6. Sellier, Alexis et al., 2017: LESS: Getting started.
<http://lesscss.org> (16.01.2017)
7. Starkov, Vladimir; Strukchinsky, Vsevolod, 2017: BEM – Block Element Modifier: Introduction.
<http://getbem.com/introduction/> (16.01.2017)
8. The Apache Software Foundation, 2017: Apache Cordova.
<https://cordova.apache.org> (18.01.2017)
9. Vontobel, Peter, 2006: Didaktisches Design aus lernpsychologischer Sicht. Pädagogische Hochschule Zürich.
http://www.scientonic.de/media/015_digimedia/050_konzepte/LIT_0210_Didaktisches_Design_Vontobel_2006.pdf (07.01.2017)

B Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

.....
Ort, Datum

.....
Unterschrift des Verfassers

C Anhang

Sämtliche Anhänge dieser Bachelorarbeit befinden sich auf der hier beigefügten CD-ROM und umfassen folgendes:

- Quellcode der App unter [Quellcode/owlisch](#) und JavaScript-Dokumentation unter [Dokumente/JavaScript-Dokumentation.pdf](#).
- Quellcode dieses Dokuments unter [Quellcode/owlisch-print](#) und PDF der Bachelorarbeit unter [Dokumente/Bachelorarbeit.pdf](#).



```
    _NL.QUIZ
    .tion() {
      lessbarAnimat
      progress();
      askQuestions();
      processQuestions();
      _slider.setSlide(_ind
      _setStep(1);
    }, (typeof event ===
```