Freie Universität Berlin

Bachelorarbeit

Erstgutachter: Prof. Lutz Prechelt

Zweitgutachterin: Prof. Claudia Müller-Birn

Entwicklung einer Desktop-Applikation zur Labororganisation mittels aktueller Webtechnologien

schriftliche Ausarbeitung

Gesa Behrends

Studiengang: BSc Informatik / Matrikelnummer: 5005892

1 Zusammenfassung

Zur Ausbildung in chemischen Studiengängen gehören üblicherweise analytische Praktika. Ihr Ziel ist es, Studierenden Analysemethoden wie chromatographische oder spektroskopische Verfahren zu vermitteln. Hierfür erhalten die Studierenden unbekannte Einzelproben und Gemische, welche zu identifizieren sind. Als Hilfsmittel, um die ausgegebenen Proben zu verwalten und zu speichern, wird in dieser Arbeit eine Informations- und Verwaltungsanwendung entwickelt.
Zunächst geht es darum, die Probenausgabe eines einzelnen Semesters zu organisieren. Längerfristig soll die Software auch dazu dienen, einen Überblick über bisher ausgegebene Proben und ihren Analyseerfolg zu erhalten, um die Lehrveranstaltung möglicherweise anzupassen.
Implementierungsseitig wird die Anwendung gänzlich mit Javascript entwickelt. Die meiste Funktionalität liegt auf der Clientseite der Anwendung, das Backend dient nur der Validierung der Daten und ihrer Persistierung.
Ein Augenmerk ist auf die Nutzerfreundlichkeit gelegt. Die Anwendung ist konservativ aufgebaut und möglichst nah an den bisherigen Abläufen, so dass sie gut in bestehende Prozesse integriert werden kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

2 Einführung

Im Rahmen dieser Arbeit wird eine Informations- und Verwaltungsanwendung entwickelt, die als Unterstützung bei der Organisation und Durchführung analytischer Praktika in chemischen Studiengängen dienen soll.

Im Rahmen solcher Praktika werden zur Einübung analytischer Techniken unbekannte einzelne Analyten sowie Analysegemische an die Studierenden ausgegeben und müssen von diesen unter Zuhilfenahme unterschiedlicher Analysemethoden identifiziert werden. Im Mittelpunkt der hier entwickelten Software soll eine geeignete Datenbank stehen, in der die zu den Praktikumsinhalten passenden Analyten sowie die bisher ausgegeben Analysegemische gespeichert werden. Die Software soll es erlauben, weitere Informationen zu den Analyten zu speichern, neue Analyten in den Praktikumskanon aufzunehmen und andere zu entfernen. Basierend auf den bisher ausgegebenen Gemischen können neue Gemische erstellt werden (randomisiert oder basierend auf den existierenden). Für die Gemische sollen bestimmte Regeln gelten: So sollte eine maximale und minimale Anzahl zu verwendender Analyten vorgegeben sein. Auch die Stoffgruppen, aus der die zu identifizierenden Chemikalien stammen, wird definiert.
Darüber hinaus soll die Anwendung auch als Informationssystem dienen. Es wird gespeichert, welche Chemikalien in welchen Semestern ausgegeben wurden. Dies kann während laufender Veranstaltung unterstützend herangezogen werden und erlaubt eine einfache Organisation. Weiterhin soll es fakultativ den Nutzern möglich sein, falsch bestimmte Proben in der Datenbank zu markieren. Dies soll den Durchführenden erlauben, Proben zu entfernen, die sich in der Vergangenheit als schwer analysierbar dargestellt haben. Optimalerweise lassen sich diese Daten durch eine geeignete deskriptive Statistik einfach abrufen.
Zuletzt muss die Software zu den ausgegebenen Proben passende Verwaltungsdokumente erstellen, dies sind eine Papierakte zur Organisation des Praktikumsbetriebs sowie Etiketten zur Probenausgabe.

Wichtig ist, dass die Software in den existierenden Arbeitsprozess passt und eine gut bedienbare Nutzeroberfläche aufweist, so dass alle wichtigen Operationen verständlich durchführbar sind.

2.1 Digitalisierung im Labor

Bei der Arbeit im Labor werden genau dokumentierte Versuche durchgeführt, anhand deren Beobachtungen und Messungen der Wissensstand erweitert werden soll.

Um diese Versuche zu reproduzieren, um sie mit Kollegen, die in thematisch benachbarten Feldern arbeiten, zu teilen und um sie final zu veröffentlichen, ist eine Informationsverwaltung essentiell. Traditionell wird hierfür ein Laborbuch eingesetzt, ein Notizbuch aus Papier, in dem handschriftlich Durchführung und Ergebnisse notiert werden. Dieses dient zunächst zur eigenen Erinnerung an durchgeführte Versuche. Es kann aber auch als Beweismittel eingesetzt werden, um Rechte am eigenen geistigen Eigentum durchzusetzen. Zuletzt dient das Laborbuch häufig auch als Archiv einer Forschungsgruppe und ist der Ort, an dem länger zurückliegende Versuche eingesehen werden können.
Insbesondere letzteres ist anhand eines handschriftlichen Dokuments schwierig. Auch wenn zu einem korrekt geführten Laborbuch ein Inhaltsverzeichnis gehört, kann die Suche nach bestimmten Versuchen dennoch zeitaufwendig sein. Insbesondere dann, wenn die Versuche länger zurückliegen und der damals experimentierende Wissenschaftler das Labor bereits verlassen hat, kann es unmöglich sein, alte Versuchsreihen zu rekapitulieren. Das Festhalten der Daten in einem elektronischen Laborbuch, das durch Vernetzung mit allen Arbeitsgruppenmitgliedern geteilt wird, etwa ist eine wichtige Entwicklung. Neben einer solchen ubiquitären Lösung werden in spezifischen Forschungszweigen auch maßgeschneiderte Lösungen entwickelt, die eine Datenintegration bieten, die genau zum Forschungsvorhaben passt.
Auch wenn die hier entwickelte Anwendung nicht die Arbeit in einem Forschungslabor unterstützen soll, ist die Zielsetzung doch die selbe: Das erworbene Wissen (in diesem Fall über ausgegebene Analyseproben) soll auch für zukünftige Nutzer*innen einfach verfügbar sein und mit den nötigen Metadaten verknüpft sein, um die zukünftige Ausgestaltung der Lehrveranstaltung zu verbessern [1, 2].

3 Erwägungen zur Usability

3.1 Mensch-Computer-Interaktion und die ISO-Norm 9241

Die entwickelte Anwendung soll als Hilfsmittel für ihre Anwender*innen dienen und ihnen die Durchführung ihrer Aufgaben erleichtern. Anforderungen an eine Software, um diese Nutzerfreundlichkeit, „Usability“ zu gewährleisten, sind neben anderen Aspekten der Interaktion zwischen Mensch und Maschine in der Norm 9241 der internationalen Organisation für Normung (ISO) standardisiert.
Eine benutzerfreundliche Anwendung ist demnach eine, die es „bestimmten Nutzern“ erlaubt in einem „bestimmten Kontext bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen“. Für eine anwenderfreundliche Software sollte also zunächst einmal die Nutzergruppe, der Anwendungskontext und das Prozessziel klar definiert sein. Einer Software wird dann in diesem Rahmen eine gute Usability zugesprochen, wenn sie dazu führt, dass die Nutzer*innen effektiv arbeiten können (also das definierte Ziel wirksam erreichen können). Außerdem soll diese Arbeit effizient sein, also mit möglichst geringem Aufwand erledigt werden können. Zuletzt sollte die Anwendung zufriedenstellend sein, die Nutzer*innen sollen also ein positives Erlebnis im Gebrauch der Software haben und ohne Störungen und Beeinträchtigungen arbeiten können.

Diese Aspekte der Wirksamkeit, der Effektivität und der Zufriedenheit der Anwender*innen werden von dem Usability-Experten Jakob Nielsen erweitert. Er schlägt insgesamt fünf Faktoren vor, die eine nutzerfreundliche Anwendung ausmachen. Neben der Effizienz und Zufriedenheit (eben wie in der ISO-Norm) sind dies für ihn Erlernbarkeit, Einprägsamkeit und die Fehlersicherheit einer Anwendung.
Eine gute Erlernbarkeit sorgt dafür, dass auch ungeübte Anwender*innen keine langwierigen Hilfestellungen konsultieren müssen, sondern möglichst schnell und einfach eingewiesen werden können. Eine einprägsam gestaltete Anwendung sorgt dafür, dass einmal erlernte Prozesse auch nach längerer Pause erneut problemlos durchgeführt werden können.
Die Fehlersicherheit gehört auch zum Bereich der Effektivität: Die Software sollte möglichst wenig fehlerhafte Handlungen ermöglichen. Falls dennoch Fehler auftreten, sollten diese durch das System abgefangen werden oder zumindest keine schlimmen Folgen haben.

Ein Begriff der etwas weiter gefasst ist als diese reine Gebrauchstauglichkeit ist die „User Experience“ (abgekürzt UX). Sie wird in Abschnitt 121 der ISO-Norm 9241

definiert. Bei der User Experience geht es um ein umfassendes Gebrauchserlebnis, es wird nicht nur der reine Nutzen betrachtet; es geht um „alle Emotionen, Vorstellungen, Vorlieben, Wahrnehmungen, physiologischen und psychologischen Reaktionen, Verhaltensweisen und Leistungen“ nicht nur bei Verwendung einer Software, sondern auch davor und danach [3].

3.2 Nutzer-zentriertes Design

Die Gestaltung und Architektur einer Software sollte unterschiedlichen Gesichtspunkte berücksichtigen. Im Mittelpunkt der Überlegungen kann etwa die Technik, theoretische Designüberlegungen, die inhaltliche Ebene oder der Nutzer stehen.
Mit einer Fokussierung auf die Technik kann zumeist schnell und sicher gearbeitet werden. Es wird der Fokus auf eine als gut bekannte Implementierung gelegt, die der/dem Entwickler*in bekannt ist; dadurch kann sicher und schnell gearbeitet werden. Alternativ wird gelegentlich auch auf eine besonders moderne Technologie geachtet, um möglichst gut auf zukünftige Weiterentwicklungen vorbereitet zu sein.
Theoretische Designüberlegungen erfolgen nach Gusto durch die Entwickler, es wird mehr oder weniger theoretisch fundiert oder nach eigenem Empfinden entwickelt.
Bei der Gestaltung nach Inhalten geben die Inhalte die Architektur vor.
Um eine gute Usability zu erhalten, liegt der Fokus beim Design sinnvollerweise beim Nutzer. Mögliche Anwender*innen werden während des Arbeitsprozesses, der Softwareunterstützung erhalten soll, beobachtet, es werden Gespräche mit ihnen geführt. Es können Prototypen gebaut werden, anhand derer die Interaktion der Nutzer mit der Software analysiert werden kann. Auch die fertige Anwendung sollte Nutzertests unterzogen werden.
Bei einem Projekt wie dem vorliegenden mit einem Fokus auf der Usability, ist ein sinnvolles Vorgehen, bei den Anwendererwartungen zu beginnen und von diesen ausgehend, das beste technische System anzustreben und umzusetzen [4].

3.3 Dialogführung und Navigation

Ein wichtiger Punkt bei der Nutzerfreundlichkeit ist die Navigation oder Dialogführung; also die Bedienfolgen, die der Nutzer durchführt, um sein Ziel zu erreichen. In der oben vorgestellten ISO-Norm 9241 werden hierzu in Teil 110 Grundsätze der Dialoggestaltung vorgeschlagen. Sie benennt als Qualitätsanforderungen Aufgabenangemessenheit, Selbstbeschreibungsfähigkeit, Steuerbarkeit, Erwartungskonformität, Fehlertoleranz, Individualisierbarkeit und Lernförderlichkeit.
Die Aufgabenangemessenheit sollte zur oben ausgeführten Effizienz und Effektivität beitragen, die Dialogführung sollte klar auf das Ziel des Nutzers ausgerichtet sein.

Ein selbstbeschreibender Dialog bietet zu jedem Schritt eine Rückmeldung oder zumindest eine Hilfefunktion, die diesen Schritt erläutert. Steuerbarkeit heißt, dass der/die Nutzer*in in der Lage ist, den Dialog mit dem Computer zu beginnen und zu steuern, sie/er kann das Tempo bestimmen und tunlichst auch die Zielsetzung. Ein guter Dialog sollte zudem den Erwartungen des/der Nutzer*in entsprechen, d.h. sie entspricht den Konventionen, wie sie aus der Ausbildung und dem Arbeitsumfeld bekannt sind. Wichtig ist auch die Fehlertoleranz, welche fordert, dass das beabsichtige Ergebnis trotz fehlerhafter Eingabe korrekt ist. Die Lernförderlichkeit meint, dass ein Dialog selber dem/der Nutzer*in beim Erlernen hilft. Zuletzt heißt Individualisierbarkeit, dass Anpassungen des Dialogs an die gestellt Aufgabe sowie die persönlichen Vorlieben des/der Nutzer*in möglich sind.

Zur konkreten Ausgestaltung der Navigation innerhalb einer Anwendung haben sich einige Konventionen etabliert: Die Nutzer erwarten, dass die wichtigsten Funktionalitäten einer Anwendung in jedem Zustand dieser über ein Hauptmenü zugänglich sind. Dieses befindet sich üblicherweise ganz oben in der Anwendung. Lokale Untermenüs werden entweder entlang der linken Seite eingeblendet (inverse L-Struktur) oder unterhalb des Hauptmenüs. Gelegentlich findet sich auch die Hauptnavigation seitlich, Unterpunkte erscheinen dann als solche innerhalb des Hauptmenüs (eingebettete Navigation).

Wichtig bei der praktischen Ausgestaltung der Navigation ist auch die sogenannte Form der Information. Ähnliche Dialogelemente müssen ähnlich gestaltet sein, es muss klar sein, wann ein Nutzer die Sicht der Seite, auf der er sich befindet, verlässt, wann nur bestimmte Teile der Seite verändert werden. Der Nutzerdialog sollte also einem klaren Muster folgen; wichtig ist auch, dass dieses Muster den Erwartungen des/der Nutzer*in entspricht.
Zusammengefasst sollte diese Anwendung eine klare Menüführung haben, die es den Nutzern immer erlaubt, schnell auf die Hauptfunktionalitäten zurückzugreifen. Es sollten wenige klar unterscheidbare Strukturelemente vorhanden sein, für die aufgrund ihrer eindeutigen Gestaltung klar ist, welchem Zweck sie dienen [4].

4 Gewählter Lösungsansatz

Im Folgenden werden die Verfahren und Implementierungsentscheidungen, die im Prozessverlauf verwendet wurden, diskutiert.

4.1 Anforderungserhebung

Wie oben erläutert, sollen bei der Entwicklung dieser Anwendung mögliche Nutzer*innen im Mittelpunkt stehen. Hier wurden die Anforderungen mittels unstrukturierter Interviews in der Arbeitsumgebung eines universitären Lehrbetriebs erhoben. Dafür wurden Nutzergespräche mit Angehörigen des Pharmazeutischen Instituts der Freien Universität Berlin geführt. Die so ermittelten Anforderungen werden als use-case-Diagramme in der Modellierungssprache UML dargestellt. Diese sind in Unterabschnitt 5.1 einzusehen. Aus diesen Anforderungen wurde eine Spezifikation erstellt, wobei komplizierte Zusammenhänge als Klassen- bzw Sequenzdiagramm ebenfalls in UML modelliert werden.

4.2 Architektur

Im Rahmen dieser Arbeit wird eine Anwendung entwickelt, in deren Mittelpunkt die Speicherung von Daten und der Zugriff auf diese steht. Solche Anwendungen werden im Kontext der Softwareentwicklung üblicherweise als CRUD-Anwendungen bezeichnet, wobei CRUD ein Akronym ist und für die grundlegenden Funktionalitäten steht, die eine solche Anwendung bieten muss: Die Speicherung (Create), das Auslesen (Read), die Aktualisierung (Update) und das Löschen (Delete) von Daten. Eine CRUD-Anwendung ist also eine, in deren Mittelpunkt (auch wenn sie durchaus weitere Funktionalität bereitstellt) eine Datenbank steht.

Hier soll die Anwendung zunächst als lokale Applikation für einen Einzelanwender entwickelt werden. Grundsätzlich wäre es allerdings nützlich, diese Anwendung für mehrere Anwender*innen an unterschiedlichen Endgeräten nutzbar zu machen. Hierfür müsste sie um einen Netzwerkasepekt erweitert werden. Um diese Möglichkeit nicht auszuschließen, sollte diese Anwendung so geplant werden, dass sie unkompliziert zu einer Webanwendung umgebaut werden kann.
Webanwendungen werden üblicherweise mittels der sogenannten Schichtenarchitektur (auch „Layers-Muster“) strukturiert. In dieser Architekturform wird die Anwendung in mehrere Schichten aufgeteilt. Tiefere Schichten sollen Dienste für höhere Schichten bereitstellen. Innerhalb einer Schicht können unterschiedliche Module da-

bei frei interagieren; zwischen den Schichten hingegen sind klare Schnittstellen definiert, die nicht variabel sind.
Eine Webanwendung hat als unterste Schicht eine Datenbank, in der die Daten, die angezeigt werden, persistiert sind. Die Datenbank wird durch eine weitere Schicht verwaltet, das Backend. Im Backend finden sich also Funktionen für die nötigen Datenbankabfragen, weiterhin werden die Daten hier häufig auf Korrektheit überprüft und es werden weitere Berechnungen und Modifikationen durchgeführt, die für den Arbeitsprozessen wichtig sind. Dies ist die sogenannte Betriebslogik. Das Backend stellt dann einen definierten Satz an Daten sowie Funktionalität zur Rückgabe von Daten für die Benutzeroberfläche bereit. Diese Nutzeroberfläche wird zumeist als Frontend bezeichnet. Es dient in erster Linie der Darstellung der Daten, kann jedoch auch Betriebslogik enthalten. Je nach verwendetem Frontend kann dieses stärker mit dem Backend verwoben sein. Die Interaktion zwischen Back- und Frontend findet über das Netzwerk statt [5].

Die Benutzeroberfläche in einer Webapplikation wird als HTML-Dokument dargestellt, welches durch CSS gestaltet und mittels Javascript animiert wird. Eine native Desktopanwendung hingegen wird mittels eines GUI-Toolkits entwickelt.
Eine native Applikation kann auf die Oberflächenelemente des Betriebssystems zugreifen und fügt sich daher nahtlos in die Umgebung ein, eine Weboberfläche kann dies nicht. Zudem kann eine native Desktopanwendung auf die Funktionalitäten des Systems zugreifen (etwa Berechnungen auf der GPU ausführen). Auf diesem Gebiet holen die Webanwendungen allerdings auf, da immer mehr Systemfunktionen mittels der Web-APIs auch über den Browser angesprochen werden können. Ein weiterer Vorteil einer nativen Desktopanwendung ist zudem die bessere Performance. Es sei hier auf das Whitepaper von Qt verwiesen (auch wenn diese als GUI-Toolkit-Entwickler natürlich parteisch sind) [6].
Trotz dieser Vorteile einer nativen Anwendung wird diese Anwendung mit Webtechnologien entwickelt. Es handelt sich um eine kleine, unkomplizierte Anwendung, so dass der Performancenachteil nicht schwer wiegt. Die Entwicklung mit Webtechnologien ist wesentlich weniger zeitaufwendig, da die Autorin mit ihr bekannten Technologien arbeiten kann und keine neuen erlernen muss. Weiterhin würde eine spätere Darstellung als Webanwendung einen echten Vorteil darstellen. Daher soll die Nutzeroberfläche hier mittels der Webtechnologien Javascript, HTML und CSS entwickelt werden.
Solange die Anwendung nur lokal und nicht über das Netzwerk abgerufen werden soll, wäre die Darstellung im Browser nicht benutzerfreundlich, stattdessen wird hier auf das Electron-Framework zurückgegeriffen. Electron bietet über einen eingebetteten Chromiumbrowser die Möglichkeit ausschließlich mit Webtechnologien

eine lokale Anwendung zu entwickeln. Eben durch diesen Browser ist jede Electron-Anwendung relativ groß und auch aufgrund der benannten Performanceprobleme häufig kritisiert. Einige bekannte Electron-Anwendungen mit großen Nutzerzahlen sind etwa Slack und Spotify (beiden sieht man das Dasein als Webclone durchaus an). Auch der insbesondere im Webumfeld beliebte Codeeditor VS Code aus dem Hause Microsoft ist eine Electron-Anwendung [7].

4.3 Datenspeicherung

Die Daten, welche in der Anwendung bereitgestellt werden, müssen persistiert werden. Es bieten sich hier unterschiedliche Verfahren an, die zunächst verglichen werden sollen, um das geeignetste Verfahren auszuwählen.
Im einfachsten Fall könnte die Persistierung Datei-basiert geschehen. Die Daten würden tabellarisiert als csv-Datei abgespeichert und über einen Dateizugriff ausgelesen und geschrieben werden. Diese Möglichkeit bietet sich bei einem einfachen Datenbestand an: Untereinander verknüpfte Daten lassen sich in einer schlichten Datei nur schwer abbilden. Ein weiterer Vor- und Nachteil ist der einfache Zugriff. Eine csv-Datei kann auch von Laien mittels üblicher Tabellenkalkulationsprogramme geöffnet und bearbeitet werden. Dies bietet auf der einen Seite die Möglichkeit eines Zugriffs auf die gesamten Daten, stellt aber auf der anderen Seite einen großen Nachteil dar: Sofern die Datei nicht mit einem Schreibschutz versehen ist, können die Daten außerhalb der vorgegebenen Programmlogik leicht modifiziert und verändert werden, so dass die Korrektheit des Programms nicht mehr sichergestellt ist. Zudem kann nicht nachvollzogen werden, welche Änderung wann durchgeführt wurde.
Eine Erweiterung dieses Ansatzes stellen die Dokumentenorientierten Datenbanken dar. Hier werden die zu speichernden Dokumente für den Zugriff mit einem eindeutigen Identifikator versehen. Die Dokumente selbst bestehen häufig aus strukturierten Daten (im XML oder JSON Format), können aber grundsätzlich jede Form von Datei enthalten. Das Datenbanksystem ermöglicht den strukturierten Zugriff auf diese Dokumente. Dokumentenorientierte Datenbanken können gewöhnlich nicht die „ACID“-Bedingungen erfüllen. ACID steht für atomicity, consistency, isolation, durability (zu deutsch Atomarität, Konsistenz, Isolation, Dauerhaftigkeit). Es sind dies die grundlegenden Voraussetzung für verlässliche Datenhaltungssysteme. Die Entwickler von Dokumentenbasierten Datenbanken legen weniger Wert auf die absolute Verlässlichkeit und Konsistenz der Daten, im Mittelpunkt steht hier vielmehr eine sehr gute Verfügbarkeit. Die Eigenschaften dieser Datenbanken werden häufig mit dem Akronym BASE bezeichnet, was für Basically Available, Soft state, Eventual Consistency stehen soll. Es ist etwas schwierig diese Begriffe ins Deutsche zu übertragen, da sie offensichtlich gewählt wurden, um ein schönes Akronympaar gegenüber

ACID zu bilden. Das Prinzip der „Eventual Consistency“ beschreibt, dass neue oder veränderte Daten bei verteilten Datenbanken nicht sofort auf alle Partitionen verteilt werden, diese Konsistenz wird zwar irgendwann (eventual) erreicht, der genaue Zeitpunkt bleibt aber ungewiss. Dieses Prinzip bietet eine gute Performance und eine einfache Skalierbarkeit.
Eine klassische relationale Datenbank hingegen stellt die Datenkonsistenz sicher, da hier die ACID-Bedingungen gewährleistet sind. Die Datenbank hat ein festes Tabellenschema, nach dem die Daten gespeichert werden. Einträge müssen also einem vorgegebenen Schema entsprechen. Wenn eine Datenreihe nicht diesem Schema entspricht und trotzdem gespeichert werden soll, muss das übergeordnete Schema angepasst werden. Dieses feste Schema bietet unterschiedliche Vorteile: Wenn die Datenbank normalisiert ist, treten keinerlei Redundanzen aus, ein bestimmter Eintrag muss tatsächlich nur an einer Stelle gespeichert werden und kann dann über Vereinigungsmethoden zu den entsprechenden anderen Einträgen hinzugefügt werden. Dieses Vorgehen verhindert Dateninkonsistenz (ein Eintrag muss tatsächlich nur einmal angepasst werden), bringt aber den Nachteil einer starken Segmentierung der Daten mit sich. Tabellen werden aufgespalten und Daten, die zusammengehören, nicht zusammen gespeichert. Eine solche Fragmentierung bringt lästige komplexe Abfragen an die Datenbank mit sich, die mehrere Tabellen vereinigen. Diese sind zum einen schwerer verständlich, bringen aber auch Performancenachteile mit sich [8].
Die Konsistenz der Daten steht bei diesem Projekt im Vordergrund, zudem sollen unterschiedliche Datenbestände (absolute Informationen wie Chemiekalientyp oder Verwendungszweck neben statistischen Werten zum Einsatz eben dieser Chemikalie) einfach vereinigt werden können, so dass für dieses Projekt eine relationale Datenbank zur Persistierung der Daten am geeignesten erscheint. Da es sich hier um ein kleines Projekt handelt, muss dabei allerdings kein Datenbankserver eingerichtet werden, der erst bei vielen nebenläufigen Anwender*innen wichtig wird. Stattdessen wird die häufig eingesetzte eingebaute Datenbank SQLite verwendet. SQLite speichert alle Tabellen in einer einzigen Datei ab und ist selber nur etwa 200 kB groß. SQLite stellt keine Typsicherheit (dass heißt fehlerhafte Werte werden dennoch als string in der Datenbank gespeichert). SQLite ist selber in C geschrieben, kann aber als natives Modul auch in NodeJS und damit in Electron eingesetzt werden [9].

4.4 Backend

Wie oben erläutert ist die Schicht, welche die Datenbank ausliest und beschreibt das Backend. Es erlaubt das Auslesen und Modifizieren der Daten, die dem/der Nutzer*in zur Verfügung gestellt werden sollen. Es erlaubt auch die Einfügung neuer

Daten in die Datenbank und sollte hierbei auf Korrektheit achten.
Die Datenbanksprache einer relationalen Datenbank ist ein Dialekt von SQL. Der einfachste Ansatz ist also, SQL-Strings direkt im Backend zu formulieren und auszuführen. Der Nachteil dieses Verfahrens ist, dass Änderungen an der Datenbankstruktur große Umstellungen des gesamten Anwendungscodes nach sich ziehen können (etwa wenn eine Tabelle aufgespalten wird). Stattdessen wird häufig an zentraler Stelle der Datenbankzugriff geregelt und dann einzelne sogenannte Modelle definiert, die Objekte in der Zielsprache darstellen und gleichzeitig den Datenbankzugriff durchführen. Diese Modelle fungieren als Objekte in der Backendlogik und können „Objekte“ in der Datenbank persistieren, indem sie ein sogenanntes Objekt-Relationales-Mapping darstellen (ORM).
Da das Frontend durch Electron dargestellt wird, ist es sinnvoll das gesamte Backend ebenfalls in Javascript zu programmieren und NodeJS als Laufzeitumgebung zu nutzen, so dass für das gesamte Projekt die selbe Programmiersprache genutzt wird.
Der Zugriff auf die SQLite Datenbank kann einfach über das SQLite-Modul erfolgen, wobei direkte SQL-Abfragen in die Programmlogik eingebaut werden müssen. Aufgrund der oben erläuterten Vorteile wird jedoch mit dem ObjectionJS-Modul mit KnexJS eine weitere Schicht oberhalb der Datenbank eingeführt. KnexJS ist ein sogenannter Query-Builder, es erlaubt die Formulierung von Migrationen, um die Datenbank systematisch anzupassen und sorgt für sichere SQL-Abfragen. Weiterhin bietet es eine einfache Anbindung an unterschiedliche Datenbanken, etwa SQLite oder Postgres. Dadurch kann die Datenbank einfach ausgetauscht werden, falls dieses Projekt später deutlich erweitert werden soll und ein Datenbankserver benötigt wird. ObjectionJS erlaubt ein einfaches Objekt-Relationales Mapping. Hier ist insbesondere von Vorteil, dass auch tiefere JSON-Objekte (object graph), die über mehrere verknüpfte Tabellen verbunden sind, einfach in die Datenbank übernommen oder ausgegeben werden können. ObjectionJS erlaubt dabei auch eine Upsert-Funktion, durch die keine unterschiedlichen Methoden aufgerufen werden müssen, um neue Datenreihen aufzunehmen: Ist ein Identifikator gegeben, wird die Reihe angepasst, andernfalls eine neue hinzugefügt. Ein weiteres populäres ORM-Modul für NodeJS ist Sequelize. Sequelize ist beliebter als ObjectionJS (500000 wöchentliche Downloads im Vergleich zu 50000 wöchentlichen Downloads von ObjectionJS), allerdings auch etwa 3.5 mal größer. Sequelize hat mehr Abhängigkeiten zu weiteren Modulen. Zudem erschien mir die Dokumentation von ObjectionJS übersichtlicher.
Zusammengefasst wird das Backend in dieser Anwendung mit NodeJS entwickelt. Zum Objekt-Mapping wird das Modul ObjectionJS eingesetzt [10, 11].

4.5 Frontend

Als oberste Schicht besteht eine CRUD-Anwendung aus der Benutzeroberfläche. Wie in Unterabschnitt 4.2 dargestellt, soll diese Oberfläche hier aus praktischen Gründen mit Webtechnologien und somit mit Javascript entwickelt werden. Als Framework wird React eingesetzt.

4.5.1 Webfrontend

Ein Webfrontend besteht zunächst nur aus einem Dokument, welches in HTML ausgezeichnet ist, mittels CSS gestaltet wird und durch interpretierten Javascript-Code animiert werden kann. Um in Seiten spezifische Inhalte einzubinden (zum Beispiel in einem Onlineshop die Produkte, die zu einer Suchanfrage passen), werden herkömmlicherweise sogenannte Templatesprachen eingesetzt. Hierbei wird in einem normalen HTML-Dokument Code als Kommentar eingebunden. Diese Codefragmente werden dann bei der Auslieferung entsprechend der aktuellen Abfrage ersetzt. Derartige Templatesprachen bieten alle üblichen Webframworks wie etwa Django (Python) oder Ruby on Rails.
Der Ersatz findet also bereits auf dem Server statt, der Browser erhält ein „schlichtes“ HTML-Dokument. Man spricht hierbei von einer Server-seitigen Rendern des Frontends. Bis auf kleinere Animationen findet sich die gesamte Logik im Backendteil der Anwendung. Templatesprachen sind (aufgrund der großen Nähe zu HTML) meist einfach zu lernen, die Performance ist nicht von der Stärke des Rechners, auf dem die Seite dargestellt wird, abhängig und bietet eine einfache Anbindung an die Datenbank. Da Suchmaschinen HTML-Seiten indexieren und ranken, werden derartige Seiten in den Suchergebnissen auftauchen.
Ein Nachteil von derartigen Templatesprachen liegt darin, dass im Prinzip ein HTML-Dokument ausgeliefert wird. Wenn daran Änderungen vorgenommen werden, muss entweder die komplette Seite neu geladen werden oder es müssen einzelne Teile der Seite mittels Javascript und gescripteten HTTP-Abfragen umgestaltet werden. Wenn Nutzer*innen also stark mit der Website interagieren, bietet es sich an der Programmierung innerhalb des Browsers mit Javascript mehr Raum zu bieten: Es wird keine Website mehr geladen, sondern vielmehr eine Webanwendung ausgeliefert. Der Server liefert zunächst nur ein rudimentäres HTML-Dokument aus, welches dann durch Javascript mit Inhalt gefüllt wird. Mittels Javascript können im Hintergrund Daten an den Server geschickt und von diesem empfangen werden, jedoch hält die Anwendung auch einen eigenen Datenbestand im Browser. Es wird so eine angenehme Benutzerinteraktion ermöglicht. Problematisch an derartigen Webanwendungen ist die Anbindung: Search engines crawlen und indexieren die HTML-Dokumente

einer Webseite, das heißt eine Javascript-Anwendung wird hier nicht gefunden. Dieses Problem lässt sich umgehen, indem das Javascript bereits auf dem Server als Templatesprache zur Generierung der „Startseite“ genutzt wird. Problematisch ist weiterhin, dass das Javascript im Browser, also auf der Hardware des/der Nutzer*in arbeitet. Bei Geräten mit schlechter Ausstattung wie günstigen Smartphones kann dies ein deutlicher Nachteil sein.

4.5.2 Laufzeitumgebung und Performance in Javascript

Javascript ist ursprünglich eine Skriptsprache, gedacht um HTML-Dokumente mit kleineren Animationen zu versehen. Wie oben dargestellt hat es in den letzten Jahren allerdings einen starken Trend zu interaktiven Seiten bis hin zu vollständigen Webanwendungen gegeben - beispielhaft sei hier für ersteres auf Twitter und andere soziale Netzwerke verwiesen und für letzteres auf die Office-Anwendung Google Docs. Die Leistungsfähigkeit solcher Anwendung liegt an der verwendeten Laufzeitumgebung. Um eine gute Performance sicherzustellen, wird in modernen Browsern Javascript daher nicht mehr ausschließlich interpretiert, sondern auch kompiliert.
Hier soll kurz das Beispiel der Javascript-Engine V8 dargestellt werden, die im Chromium-Browser (der Rendering-Engine von Electron) eingesetzt wird. Auch die backendseitige Javascript-Laufzeitumgebung NodeJS basiert auf V8. V8 besteht aus dem Bytecode-Interpreter Ignition und dem optimierenden Compiler Turbofan. Während Ignition Javascript in Bytecode übersetzt, sammelt es weitere Daten. Wenn ein Programmfragment häufig verwendet wird, wird es mit den gesammelten Daten (etwa verwendete Typen einer Variablen) an Turbofan übergeben, welches den Bytecode in optimierten Maschinencode übersetzt. [12]

4.5.3 React

Die oben angesprochenen Webanwendungen werden aktuell häufig mit Webframeworks entwickelt, die den Bau eines Komponenten-basierten Frontends erlauben. AM verbreitesten sind Angular, Vue und React. In diesem Projekt wird React eingesetzt. Dieses für sich genommen ist kein echtes Framework, sondern sind zunächst nur zwei Javascript-Bibliotheken, welche es erlauben das virtuelle DOM zu verwalten und DOM-Knoten aus Javascript heraus zu generieren. Das DOM (Dokument-Objekt-Modell) ist die Programmierschnittstelle, die es erlaubt, die Elemente eines HTML-Dokuments anzusprechen und zu modifizieren. React kann also dazu genutzt werden, mittels Javascript DOM-Knoten zu generieren oder anzupassen. Ein einfaches Beispiel ist:

```
const HelloWorld = React.createElement(
        "p", null, "Hello World!"
)
```

React erzeugt hier einen DOM-Knoten vom Typ p, das als Kind den String `"Hello World!"` erhält. Im Parameter an zweiter Stelle können weitere Eigenschaften des DOM-Knotens definiert werden, wie etwas die Klasse zur Gestaltung durch CSS, eine id oder Eventhandler. Durch Aufruf der `render`-Funktion des ReactDOM wird das Element dem React-DOM hinzugefügt, welcher die Interaktion mit dem tatsächlichen DOM übernimmt. In der Webanwendung findet sich ein einfaches Paragraph-Element:

```
<p>Hello World!</p>
```

Zur Vereinfachung gibt es eine üblicherweise in der React-Programmierung eingesetzte Syntax, die HTLM ähnelt: Javascript XML erlaubt es innerhalb des Javascript-Codes DOM-Knoten in der render-Funktion zu benennen. JSX wird über einen Transpiler dann in die `React.createElement` Methode übersetzt. Das gleiche funktioniert auch mit React-Komponenten. Die oben benannte HelloWorld-Funktion kann damit als `< HelloWorld />` dargestellt werden.
Die zweite grundlegende Bibliothek ist ReactDOM. Diese dient dem performanten Rendern der Reactkomponenten im Browser-DOM. ReactDOM erlaubt es, eine Kopie des aktuellen Zustands des DOMs im Speicher zu halten. Wenn an der Benutzeroberfläche Veränderungen vorgenommen werden, wird genau errechnet, welche Knoten zu verändern sind und spezifisch nur diese angepasst. Dies bietet den weiteren Vorteil, dass der/die Programmierende nicht mehr selber auf das DOM zugreifen muss, sondern dieses möglichst effektiv durch React vermittelt wird [13].

Weitere Funktionalitäten, welche nicht zu React dazu gehören, können als zusätzliche Module integriert werden. Zu nennen wären hier ein Javascript-Router, der eine Navigation innerhalb einer Webanwendung über die URL erlaubt. Bei Änderung der URL wird kein neues Dokument vom Server angefordert, stattdessen wird der Sichtwechsel innerhalb der Javascriptanwendung, d.h. clientseitig durchgeführt. Für React wird der React-Router eingesetzt. Häufig eingesetzt werden zudem Module zum sogenannten State-Management, also zur Datenverwaltung im Frontend. Es erlaubt dieses sämtlichen Komponenten einen einfachen Zugriff auf die in der Anwendung verwendeten Daten, ohne selbst beständig neue eigene HTTP-Anfragen zu generieren. Eine häufig verwendete Lösung im React-Umfeld ist die library redux [14].

4.5.4 Webpack als Modulpacker

Aus einer einfachen Webseite, einem HTML-Dokument mit etwas css und Javascript, ist bei der Arbeit mit einem Webframework eine komplexe Anwendung geworden. Es sind nicht mehr nur drei Dateien, sondern alleine für das Javascript gewöhnlich so viele Dateien, wie unterschiedliche Komponenten eingesetzt werden. Jede dieser Komponenten hat zudem häufig eine eigene css-Datei, die das Aussehen definiert. Um nicht zahllose script-tags einbauen zu müssen, werden alle diese Dateien zur Auslieferung zusammengefasst. Dafür wird ein sogenannter Modulpacker („module bundler“) eingesetzt. Verbreitete Modulpacker sind Browserify und Webpack. In diesem Projekt wird Webpack eingesetzt, daher soll hier auf dessen Funktionsweise eingegangen werden.
Zur Zusammenfassung aller in einem Projekt erforderlichen Komponenten muss zunächst ein Einstiegspunkt definiert werden (üblicherweise als index.js bezeichnet). Von diesem Einstiegspunkt analysiert Webpack aller weiteren erforderlichen Komponenten, die über `import` bzw `require` in das Projekt eingebunden sind. Hieraus wird der sogenannte Abhängigkeitsgraph der zu packenden Anwendung erstellt; alle in diesem Graph verwendeten Module werden dann in einem Bundle, üblicherweise in einer einzigen Datei zusammengefasst. Ähnlich wie React ist auch Webpack stark modularisiert, es können zahlreiche weitere Plugins eingesetzt werden. Die in diesem Projekt verwendeten sollen hier erläutert werden.
Wenn, wie oben dargestellt, mehrere CSS-Dateien angelegt werden, müssen diese analog zu den Javascriptmodulen zu einer einzigen Datei zusammengefasst werden. Hier wird dafür der post-css-loader eingesetzt. Auch post-css selber kann wiederum um diverse passende Plugins erweitert werden. Hier wird das advanced-variables plugin eingesetzt, welches es einfach erlaubt, Variablen zu definieren, die global verwendet werden können (ohne in jeden Stylesheet extra importiert zu werden). Außerdem findet das nested-Plugin Verwendung, welches die Verschachtelung von css-Selektoren erlaubt und so für vereinfachte Stylesheets sorgt.
Zudem wird Webpack zum Transpilieren eingesetzt. Transpilieren heißt zunächst die Übersetzung von einer Programmiersprache in eine andere. Insbesondere bei Javascript tritt dies häufig auf: Zum einen wird gelegentlich auch das Frontend in der Backendsprache geschrieben und muss dann über einen Transpiler in das vom Browser interpretierbare Javascript übersetzt werden. Zum anderen gibt es aufgrund der Javascript inhärenten Probleme wie fehlende Typisierung diverse Varietäten wie Coffeescript oder Typescript, die dieses entschärfen sollen. Browser verstehen jedoch nur die in den ECMA-Spezifikationen festgeschriebenen Sprachkonstrukte. Alle anderen Javascript-Varianten müssen also vor der Auslieferung in Standardjavascript übersetzt werden. Selbst bei der Verwendung von standardisiertem Javascript muss

häufig ein Transpiler eingesetzt werden, weil moderne Funktionalitäten der Sprache (etwa die Pfeilfunktionen, die funktionales Programmieren in Javascript erlauben) bereits standardisiert wurden, aber noch nicht von allen Browserherstellern in ihre Javascript-Engines übernommen wurden. Modernes Javascript muss also zunächst in allgemein verständliches Javascript übersetzt werden. Auch bei der Verwendung von React mit JSX muss immer ein Transpiler eingesetzt werden. Dieser übersetzt die JSX-Syntax in die oben aufgeführte `React.createElement()`-Funktion. Dieses geschieht über den Transpiler Babel.
Bei der Entwicklung einer Webanwendung sollten die zu verschickenden Dateien möglichst klein sein, um Bandbreite zu sparen. Dafür werden Whitespace und Kommentare entfernt und Variablennamen angepasst. Dieser Prozess wird minification genannt und kann ebenfalls mittels Plugin durch Webpack erfolgen.
Zuletzt sei das Hot-Module-Replacement genannt, durch welches Webpack die Entwicklung einer Webanwendung beschleunigt: Im Rahmen der Entwicklung einer modularen Webanwendung müssen einzelne Module angepasst oder ausgetauscht werden. Um diese Veränderung sichtbar zu machen, müsste gewöhnlich die gesamte Anwendung neu gebündelt und dann ausgeliefert werden. Das Hot-Module-Replacement erlaubt es, nur die veränderten Module neu zu laden [15].

4.6 Entwicklungsumgebung

An Universitäten werden üblicherweise Microsoft-Betriebssysteme eingesetzt, daher wird in einer Windows-10-Umgebung entwickelt. Als Entwicklungsumgebung wird Visual Studio Code eingesetzt. Hier wird das ESLinter-Plugin verwendet, um den Code statisch zu analysieren und anzupassen.
Der Code wird auf github gehostet. Zur Visualisierung der git-history wird Sourcetree eingesetzt.
Zur Visualisierung von Anwendungsfällen und Klassenzusammenhängen (Anwendungsfalldiagramm und Klassendiagramm) gemäß der UML-Spezifikation wird das Latex-Paket `metauml` eingesetzt. Zur Darstellung des Sequenzdiagramms das Latex-Paket `pgf-umlsd`.

5 Dokumentation der Durchführung und der entstandenen Artefakte

5.1 Anforderungen

Die Anwendungsfälle wurden in unstrukturierten Interviews mit möglichen Anwender*innen der Software am Pharmazeutischen Institut der Freien Universität Berlin ermittelt. Es wurden dabei drei unterschiedliche Anwendungsfälle analysiert, welche in Abbildung 1 dargestellt sind: Die Software muss es ermöglichen, für ein neues Semester einen Satz Proben zu generieren, der den durch den/die Anwender*in definierten Anforderungen entspricht. Es soll hier möglich sein, die Teilnehmeranzahl anzupassen, sowie die Art von Proben, die ausgegeben werden. Nach Erstellung dieser Proben soll die Software einen schnellen Überblick über das laufende Semester geben (welche Proben wurden ausgegeben). Auch auf weiter zurückliegende Semester soll zugegriffen werden können. Zuletzt soll die Anwendung einen statistischen Überblick über die bisher durchgeführten Praktika erlauben. Der wichtigste Anwendungsfall ist dabei die Erstellung eines Probensatzes für ein neues Semester. Er ist detailliert in Abbildung 2 dargestellt.

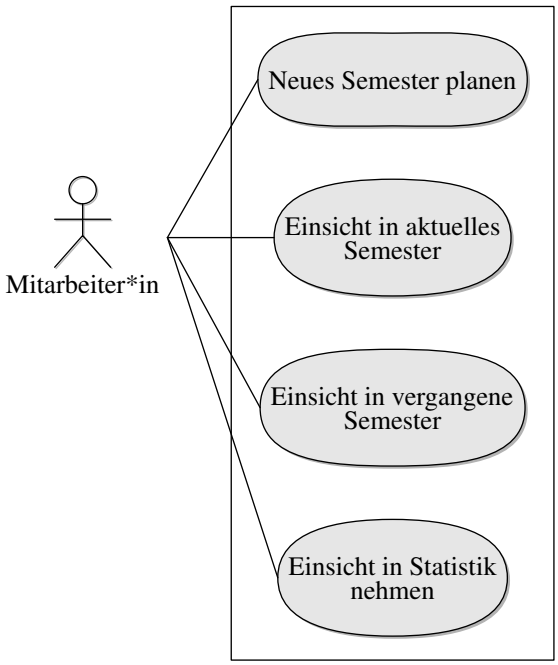

Abbildung 1: **Anwendungsfälle der entwickelten Anwendung:** Die Anwendung muss es ermöglichen, Probensätze für ein neues Semester zu generieren, auf die im aktuellen Semester verwendeten Proben zurückzugreifen, sowie sollte einen Einblick in die bisher ausgegebenen Proben erlauben

Ein*e Mitarbeiter*in möchte Proben für ein neues Semester generieren. Es ist definiert, wieviele Proben welcher Zusammensetzung für welche analytische Methodik benötigt werden. Die zu entwickelnde Anwendung stellt diese Proben (unter Zuhilfenahme der Datenbank) bereit und speichert diese, zwecks späterer Auswertung automatisiert ab. Diese Anwendungsfall beinhaltet mögliche Erweiterungen: Zu Semesterbeginn ist es ebenfalls üblich, Änderungen am Datenbestand vorzunehmen. Das heißt es können nun ebenfalls die Anwendungsfälle Analyt entfernen bzw neu hinzufügen sowie Austausch desselben in Gemischen durchgeführt werden. Im Folgenden soll die Anwendung es dem/der Nutzer*in erlauben, eine Papierakte zur Verwaltung zu erzeugen, sowie Probenetiketten, die bei der praktischen Durchführung im Labor benötigt werden. Die weiteren Anwendungsfälle „Überblick über das aktuelle Semester“, „Überblick über vergangene Semester“ sowie „Einblick in die Statistik“ haben einen einfachen Ablauf und sind daher hier nicht weiter graphisch dargestellt.

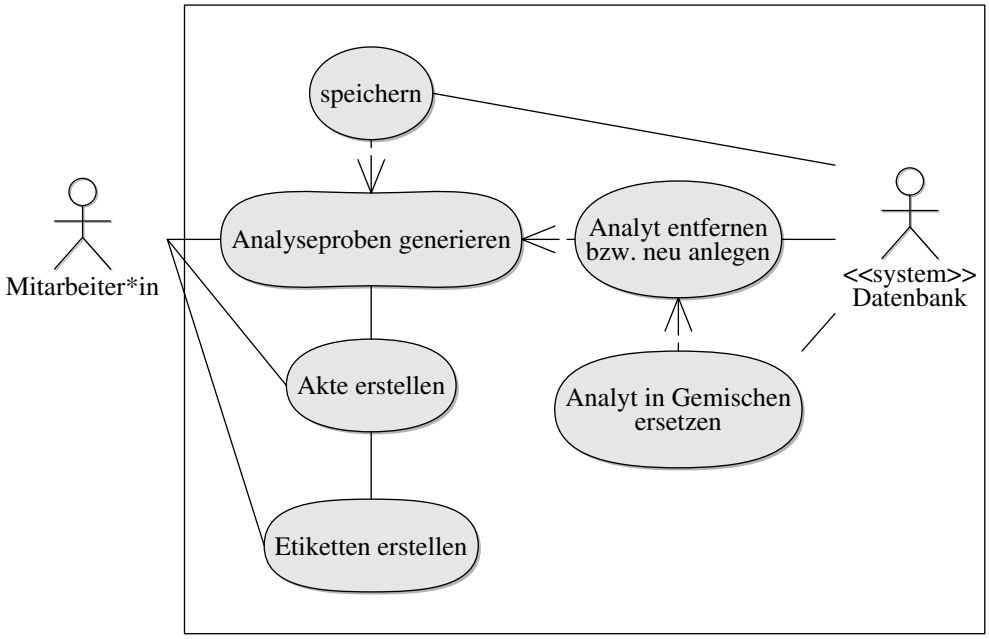

Abbildung 2: **Wichtigster Anwendungsfall der entwickelten Anwendung: Generierung der Proben für ein neues Semester.** Ein*e Nutzer*in der Software generiert neue Analyseproben für ein neues Semester. Die Proben werden samt Verwendungsjahr in der Datenbank gespeichert, entsprechende Verwaltungsdokumente generiert.

5.2 Datenmodellierung

Die entwickelte Anwendung muss sieben unterschiedliche Datenobjekte speichern: Zum einen die Definition der Analytenklassen und der Gemischklassen sowie die tatsächlich verwendeten Analyten und Gemische. Daneben müssen einzelne Semester

samt den ausgegeben Proben verwaltet werden. Die notwendigen Attribute sowie Relationen zwischen diesen Objekten werden hier erläutert.

Zunächst muss die Datenbank sozusagen eine Blaupause der auszurichtenden Veranstaltung enthalten. Die Nutzenden geben an, welche Stoffgruppen als Analyten eingesetzt werden, und wie die einzelnen Gemische aus diesen zusammengestellt werden sollen. Diese Klassen sind als Analytenklasse bzw Gemischklasse bezeichnet und sind in der oberen Hälfte von Abbildung 3 dargestellt. Sie benötigen je einen eindeutigen Identifikator und zusätzlich einen den Anwender*innen verständlichen Namen.

Weiterhin werden in der Datenbank alle Chemikalien, die als Analyten eingesetzt werden, sowie alle Gemische, die bisher verwendet wurden, verwaltet. Die grundlegende Modellierung dieser Daten ist in der unteren Hälfte von Abbildung 3 dargestellt. Wichtige Attribute sind für beide ein eindeutiger Identifikator, sowie ein verständlicher Name (bei den Chemikalien bietet sich hier der üblichste Trivialname an). Beide sollen auch aus dem Praktikumskanon entfernt werden können, dabei allerdings weiterhin in der Datenbank gehalten werden. Hierfür erhalten sie jeweils ein Flag namens isActive und enstprechende Setter-Methoden. Der Analyt benötigt weiterhin ein Attribut, welches auf die Stoffklasse verweist, das Gemisch, die Möglichkeit einen Analyten auszutauschen / zu entfernen. Ebenfalls zur Erhaltung einer sinnvollen Statistik sollte diese Methode allerdings nicht tatsächlich auf ein Gemisch angewandt werden, stattdessen sollte das ursprüngliche Gemisch inaktiviert werden (so dass es nicht mehr aktiv Verwendung findet) und ein neues hinzugefügt werden, um keine Informationen zu verlieren. Beide Klassen könnten sinnvoll durch weitere Attribute ergänzt werden (etwa die CAS-Nummer für die Analyten).

Der nächste wichtige Zusammenhang ist der zwischen einem Semester, den verwendeten Proben und auf welchem Gemisch diese beruhen. Er ist in Abbildung 4 dargestellt. Jedes Semester wird das Praktikum von mehreren Studierenden besucht. Für jede*n wird ein Probesatz vorgehalten, welcher einen eindeutigen Identifikator in der Datenbank sowie einen im Praktikumsbetrieb leichter zu verwendenden Codenamen (etwa fortlaufend im Semester) erhält. Ein Probensatz besteht aus mehreren anonymisierten Proben, ebenfalls mit Codenamen und Identifikator. Die Proben erhalten ein Boolsches Flag, so dass diejenigen, welche nicht erfolgreich analysiert wurden, markiert werden können.

5.3 Ablauf bei der Erstellung eines neuen Semesters

Wie oben ausgeführt ist der wichtigste Anwendungsfall die Generierung der Proben für ein neues Semester. Die Abläufe, die hierfür in der Anwendung durchlaufen

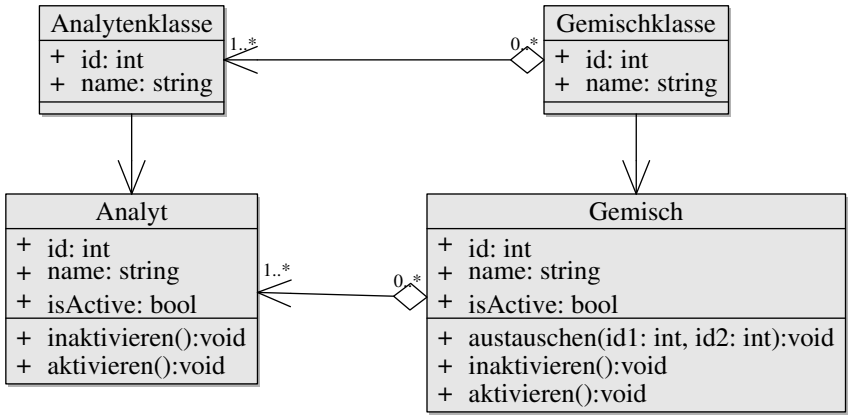

Abbildung 3: **Klassendiagramm zur Darstellung der Beziehungen zwischen den Analyten und ausgegebenen Probegemischen sowie ihren Vorgaben.** Die Datenbank enthält die Klassen „Analytenklasse“ sowie „Gemischklasse“, in denen die Zusammenstellung unterschiedlicher Proben gespeichert ist. Diese Klassen dienen als Bauplan für konkrete Analyten und Gemische. Analyten sind sämtliche Chemikalien, die zur Analyse ausgegeben werden können, Gemische konkrete Zusammenstellungen dieser zu Analytengemischen.

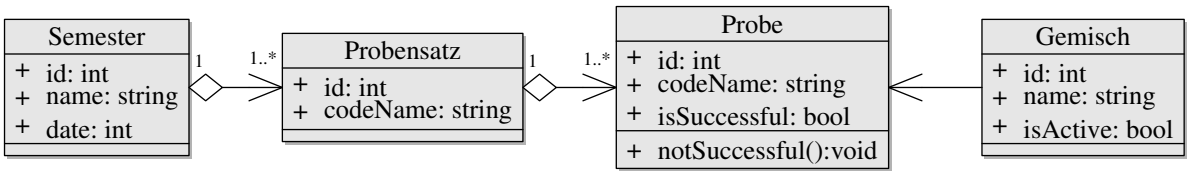

Abbildung 4: **Klassendiagramm zur Darstellung der Klassen, welche bei Generierung eines neuen Semesters angelegt werden.** Jedes Semester besteht aus so vielen Probesätzen, wie Studierende teilnehmen. Jeder Probensatz setzt sich aus vorgegebenen Gemischen zusammen, diese werden mit einem Codenamen abgespeichert.

werden, sollen im Folgenden skizziert und ist in Abbildung 5 dargestellt. Auf die erweiternden Fälle „Entfernen eines Analyten“ und „Ersatz eines Analyten in bestimmten Gemischen“ wird noch nicht eingegangen.

Der/Die Nutzer*in möchte ein neues Semester beginnen und benötigt zufällig generierte Probensätze einer bestimmten Anzahl und Art. Beides wird vom Semestermodul abgefragt. Welches die entsprechenden Proben, welches die entsprechenden Proben dann vom Backend anfordert. Das Backend erfragt diese Proben aus der Datenbank, es erzeugt zusätzlich einen CodeNamen für jede Probe und jeden Probensatz (es sei auf das Datenbankschema in Abbildung 4 verwiesen). Die Codenamen werden samt dem aktuellen Semester und den zugehörigen Gemischen in der Datenbank gespeichert. Die generierten Proben werden an die Benutzeroberfläche zurückgegeben, hier werden nun unabhängig voneinander ein pdf-Dokument für die Verwaltungsakten und eines für die Laboretiketten generiert. Im Sinne einer guten Nutzbarkeit sollten die Einzelaktionen jeweils durch den/die Nutzer*in gesteuert

werden können und nicht automatisiert ablaufen.

5.4 Ordnerstruktur des Projekts

Es gibt für NodeJS keine Standardstrukturierung wie sie etablierte Frameworks wie Ruby on Rails oder python django bieten. Für dieses Projekt wurde die in Abbildung 6 dargestellte Ordnerstruktur eingesetzt.

Die rc-Files definieren zum einen die Konfiguration des Javascript-Linters (eslint.rc), zum anderen die Konfiguration des Transpilers. Der Javascript-Linter basiert auf den Standardregeln, die durch das Entwicklerteam von Airbnb festgelegt wurden. Der Knexfile gibt Informationen über die Datenbank und ist unterteilt in Produktions- und Entwicklungsumgebung. Dies ist insbesondere hier hilfreich, da durch den electron-packager, die Datenbank als asset in einen anderen Ordner verschoben wird. Das package.json enthält neben den verwendeten nodeJS-Modulen einfache Skripte, um die Anwendung im Entwicklungsmodus zu starten und sie zu Produktionszwecken zu verpacken. In der Webpackconfig werden die unterschiedlichen Webpack-Plugins aufgerufen, es wird definiert, von wo der Code kompiliert werden soll. Wichtig ist auch hier die Unterscheidung zwischen Entwicklungs- und Produktionsumgebung. In der Entwicklungsumgebung startet Webpack einen eigenen Webserver, der das Electronfenster beliefert. Dieses ist notwendig, um das Entwickler-freundliche oben erwähnte Hot Module Replacement zu erlauben.
Im dist-Folder befindet sich der gebündelte Code für die Produktionsumgebung, der Datenbankfolder beinhaltet die Modelle, Controller die das Backend bilden, sowie die Datenbankmigrationen. Im src-Folder befindet sich der gesamte Clientcode inklusive der css-Stylesheets. Der Ordner ist grob untergliedert in `views`, hier finden sich jeweilige ganzseitige Nutzeransichten und `components`, hier finden sich die kleineren Komponenten, aus denen die Nutzeransichten zusammengebaut werden. Im `functions`-Ordner befindet sich die Javascriptlogik, etwa zum Generieren der pdfs. Die `main.js`-Datei ist der Hauptprozess der Electronanwendung, hier werden die Browserfenster generiert, in denen die Benutzeroberfläche gerendert wird. Zudem kann dieser Prozess auf sämtliche native Funktionen des Betriebssystems zugreifen.

5.5 Gestaltung des Backends

Diese Applikation soll in erster Linie clientseitig laufen, das Backend ist nur noch zur finalen Validierung der Daten und des Aufrufs klassischer Datenbankfunktionen der Speicherung, des Auslesens, der Aktualisierung und des Löschens von Daten zuständig. Ein Modell benötigen in erste Linie die Klassen Analyt, Gemisch sowie

Semester. Hier werden im Model tatsächlich nur die Relationen und die Attribute, die die entsprechende Tabelle enthält, festgehalten. Die Datenbankabfragen selber sind in den entsprechenden Controllern festgeschrieben.
Diskussionswürdig ist zunächst die Abfrage einer vorgegebenen Anzahl zufälliger Proben eines bestimmten Typs im Controller zum Gemisch Modell.

```
Gemisch
        .query()
        .orderBy(raw('random()'))
        .limit(number)
        .eager('[...]');
```

Zur Generierung zufälliger Proben wird hier auf die `random()`-Funktionen zugegriffen, die Anzahl benötigter Proben kann als `number` an die `limit`-Bedingung übergeben werden. `Eager` ist ein Schlüsselwort des ORM objectionJS und erlaubt es weitere definierte Relationen anzufügen. Problematisch an dieser Stelle ist die Nutzung der OrderBy-Methode, die dafür sorgt, dass die gesamte Tabelle sortiert wird. Bei einem Projekt wie diesem, bei dem sich die Anzahl der Reihen noch im Größenbereich der Tausender bewegt ist dieses unkritisch. Sollte das Projekt jedoch deutlich vergrößert werden, sollte an dieser Stelle auf effektiveren Code zurückgegriffen werden, beispielsweise in dem für numerische Identifikatoren (wie sie in diesem Projekt Verwendung finden) eine passende Zufallszahl im Programm generiert und die entsprechende Reihe aus der Datenbank abegerufen wird. (Bei durch delete entfernten Datenbankeinträgen könnte entweder neu gestartet werden oder der nächste Eintrag nach dem Loch ausgewählt werden - problematisch natürlich bei großen Löchern). Interessant ist zudem der Code zur Generierung von eindeutigen Zufallsidentifikatoren der auszugebenden Proben; diese folgen einem Muster, um die Arbeit im Labor zu erleichtern. Sie bestehen immer aus einem Buchstaben gefolgt von einer dreistelligen Ziffernkombination. Die pro Semester ausgegebenen Proben erhalten dabei den gleichen Buchstaben. Es ist möglich sein, den Buchstaben auszuwählen, es kann aber auch ein zufälliger Buchstabe durch die Anwendung zugewiesen werden. In beiden Fällen muss sichergestellt sein, dass dieser Buchstabe in den letzten zehn Jahren nicht verwendet wurde, so dass es zu keinen Uneindeutigkeiten im Labor kommt.

5.6 Gestaltung der Navigation

Wie in Abschnitt 2 ausgeführt ist die Navigationsführung essentiell für eine gute Nutzbarkeit einer Anwendung. Hier gibt es vier unterschiedliche Hauptanwendungsfälle der Anwendung („neues Semester“, „Überblick über das aktuelle Semester“, „Überblick über vergangene Semester“ sowie „Einblick in die Statistik“, vgl. Ab-

bildung 1). Für eine übersichtliche Navigation erhält jeder eine eigene Sicht. Die erweiternden Anwendungsfälle aus Abbildung 2 werden hier unter „Anpassen des Datenbestandes“ in einer Sicht zusammengefasst. Damit besteht die Anwendung aus 5 Sichten, die vom Startbildschirm, dem Dashboard, aus zugänglich sind. Zusätzlich gibt es eine Querverbindung von der Generierung eines neuen Semesters hin zur Anpassung des Datenbestandes (vgl auch Abbildung 2). Die Navigation ist in Abbildung 7 dargestellt.

Alle fünf Hauptsichten sind auch von einem Menü aus zugänglich. Dieses wird an der Oberkante der Anwendung dargestellt. Die Sichten mit komplexeren Eingabemasken („neues Semester“, „Anpassen des Datenbestandes“) zeigen vor einem Sichtwechsel eine alert-Nachricht an, um vor dem Verlust der eingegebenen Daten zu warnen.

5.7 Pdf-Generierung

Wichtiger Bestandteil des Projekts ist die Ausgabe von gedruckten Artefakten, damit die bisher üblichen Prozesse weiter verfolgt werden können. Dieses könnte backendseitig über die Anbindung an eine Office-Bibliothek oder eine Tex-Distribution erfolgen, wird hier allerdings mit dem einfachen Javascript-Modul jsPdf gelöst. Dieses erlaubt weniger Gestaltungsfreiraum, reicht hier allerdings aus. Es sind kleinere Anpassungen in der Gestaltung möglich (z.B. Änderung der Etikettengröße). Zudem ist wichtig, dass die Akte über eine größere Bandbreite an möglichen Proben eine ansprechende Optik behält. Hierfür wird die Seite komponentenweise zusammengesetzt, so dass Anpassungen möglich sind.

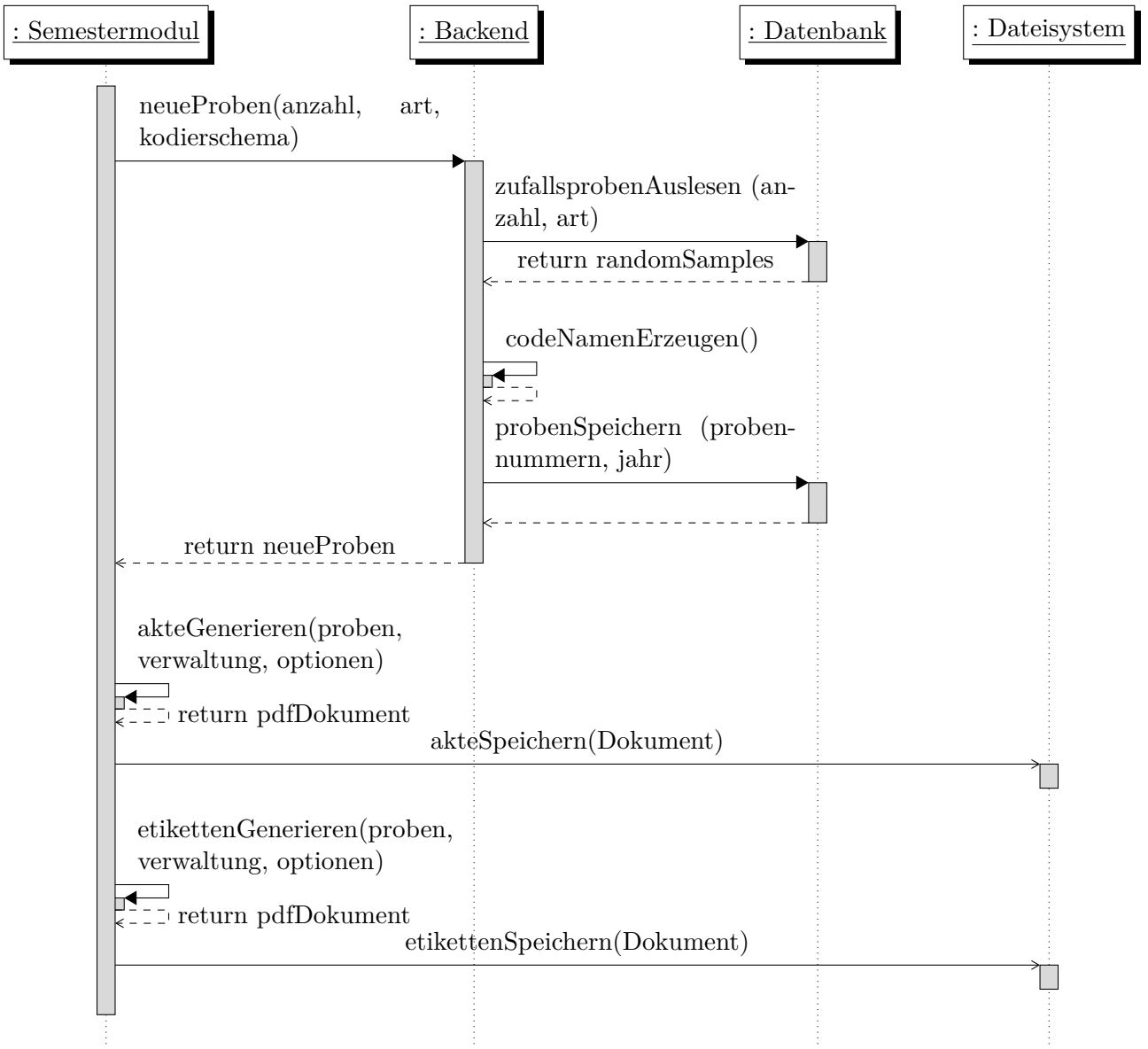

Abbildung 5: **Abfolge der Funktionsaufrufe im Modul zur Generierung der Proben eines neuen Semesters**

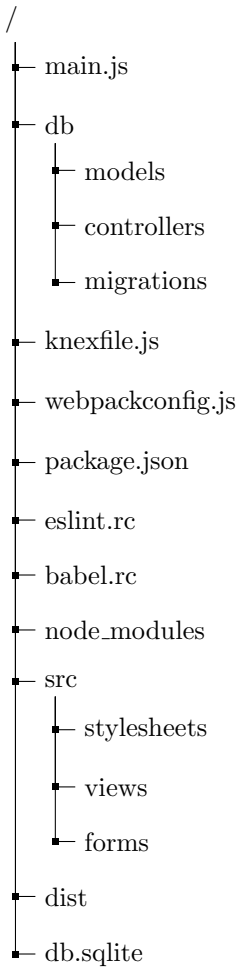

Abbildung 6: **Die in diesem Projekt verwendete Ordnerstruktur.** Der db-Ordner enthält die Backend-Module, src die zu bündelnden Frontendmodule.

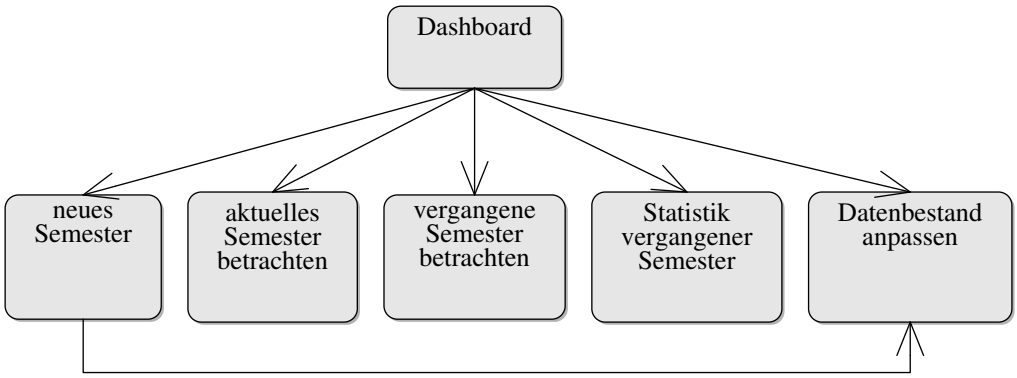

Abbildung 7: **Die in diesem Projekt verwendeten Sichten**. Alle Sichten sind vom Dashboard sowie dem Hauptmenü aus zugänglich. Zusätzlich gibt es eine Querverbindung von der Generierung eines neuen Semesters hin zur Anpassung des Datenbestandes.

6 Ausblick

Im Rahmen dieser Arbeit wurde eine CRUD-Applikation entwickelt, die es erlaubt, den Probenbestand eines analytischen Praktikums zu verwalten und basierend auf bisherigen Durchgängen Probensätze für neue Semester zu generieren. Es ist dem/-der Anwender*in möglich, neue Probenklassen in den Kanon aufzunehmen und neue Analyten einzusetzen. Mit der Anwendung können die Dokumente, die zur Verwaltung des Praktikums notwendig sind, erzeugt werden. Zudem erlaubt ein Statistikmodul einen Einblick in die ausgegebenen Proben zu nehmen, um Verbesserungspotential einzusehen.
In nächster Zeit sollte die Anwendung noch einem Nutzertest unterzogen werden, um die Konzeption der graphischen Oberfläche in der tatsächlichen Anwendung zu untersuchen. Funktional könnte die Applikation erweitert werden: Das einfache Statistikmodul könnte ausgebaut werden, etwa um neue Probenkombinationen vorschlagen, die zu den bekannten Proben passen. Es wäre denkbar, die Anwendung als Netzanwendung umzugestalten, so dass sie sowohl im Vorlauf aus dem Büro, während der Erstellung der Proben im Labor und bei der Leistungskontrolle während der Durchführung eingesetzt werden könnte. Der Probenerfolg müsste nicht mehr nachträglich eingetragen werden, sondern könnte direkt aus dem Labor in die Datenbank übernommen werden. Es wäre allen Mitarbeitern transparent möglich, den Fortschritt des Praktikums zu verfolgen.

Literatur

[1] Martin Kühne und Andreas W. Liehr. „Improving the Traditional Information Management in Natural Sciences“. eng. In: *Data Science Journal* 8.0 (2009), S. 18–26. ISSN: 1683-1470.

[2] Florian Hauer. „Vom Papier auf die Plattform“. In: *Nachrichten aus der Chemie* 61.12 (2013), S. 1234–1235.

[3] Jakob Nielsen. *Usability engineering*. eng. Boston [u.a.]: Acad. Press, 1993. ISBN: 0-12-518405-0. URL: https://primo.fu-berlin.de/FUB:FUB_ALMA_DS21840814390002883.

[4] James Kalbach. *Designing Web Navigation*. en. Beijing [u.a.]: O'Reilly, 2007. URL: https://www.oreilly.com/library/view/designing-web-navigation/9780596528102/.

[5] Oliver Vogel u. a. „Software-Architektur: Grundlagen — Konzepte — Praxis“. ger. In: 2. Auflage. Heidelberg: Spektrum Akademischer Verlag, 2009. ISBN: 9783827419330.

[6] Burkhard Stubert. *Qt or HTML5?* 2017. URL: https://www.qt.io/html (besucht am 29. 03. 2019).

[7] *Electron, cross platform desktop apps with JavaScript, HTML, and CSS*. URL: https://electronjs.org/ (besucht am 23. 03. 2019).

[8] Michael Stonebraker. „SQL databases v. NoSQL databases“. eng. In: *Communications of the ACM* 53.4 (2010), S. 10–11. ISSN: 1557-7317.

[9] *SQLite*. URL: https://www.sqlite.org/index.html (besucht am 23. 03. 2019).

[10] *npm*. URL: https://www.npmjs.com/ (besucht am 23. 03. 2019).

[11] *ObjectionJS*. URL: https://vincit.github.io/objection.js/ (besucht am 23. 03. 2019).

[12] v8 Team. *Launching Ignition and TurboFan*. 2017. URL: https://v8.dev/blog/launching-ignition-and-turbofan (besucht am 10. 03. 2019).

[13] Facebook Inc. *Virtual DOM and Internals*. 2019. URL: https://reactjs.org/docs/faq-internals.html (besucht am 15. 03. 2019).

[14] Alex Banks und Eve Porcello. *Learning React: Functional Web Development with React and Redux*. 1st. O'Reilly Media, Inc., 2017. ISBN: 1491954620, 9781491954621.

[15] *Webpack*. URL: https://webpack.js.org/ (besucht am 22. 03. 2019).