Dokumentation zum Praktikum XML-Technologie

Group XSLT

Table of Contents

1. Gliederung und Beschreibung der Ansichten

Das Spiel ist logisch in 3 Ebenen unterteilt. Zunächst eine Anmeldeseite, auf welcher Spielern die Möglichkeit gegeben wird, sich mit Benutzername und Passwort anzumelden. Wurde noch kein Account mit den angegebenen Anmeldedaten erstellt, so wird dieser mit den Eingabewerten angelegt. Nach einem Klick auf Login wird der Spieler weitergeleitet zur Lounge-Ansicht, in der Spielern offene Spiele angezeigt werden, sowie eine Highscore-Liste. Die Lounge bietet die Möglichkeit, ein neues Spiel zu starten oder ein gespeichertes Spiel zu laden und fortzusetzen. Beim Start eines neuen Spiels, kann der Spieler auswählen, mit wie vielen Kartenpaaren, und Personen er spielen möchte. Jedem Mitspieler kann zudem ein Name gegeben werden. Sobald ein Spiel gestartet wurde, wird die zweite Ebene, ein Spielfeld mit der gewünschten Kartenanzahl angezeigt, wobei die Karten zu beginn alle verdeckt liegen, wenn es sich um ein neu gestartetes Spiel handelt. Zudem werden die Spieler mit ihren aktuellen Punkten angezeigt, wobei der aktuelle Spieler farblich hervorgehoben ist. Während des Spielens, wird diese Benutzeroberfläche nicht verlassen. Die Interaktionsmöglichkeiten in dieser Ansicht sind das Umdecken von Karten sowie die manuelle Spielunterbrechung. Diese ist zu jedem Zeitpunkt möglich. Dabei kehrt das Spiel wieder in die Lounge-Ansicht zurück. Das Spiel ist beendet, sobald alle Karten aufgedeckt wurden. Daraufhin wird anstelle der Karten eine Benutzeroberfläche mit dem Ergebnis des Spiels angezeigt. Hier ist es wieder möglich ein neues Spiel aus der Lounge-Ansicht zu starten.

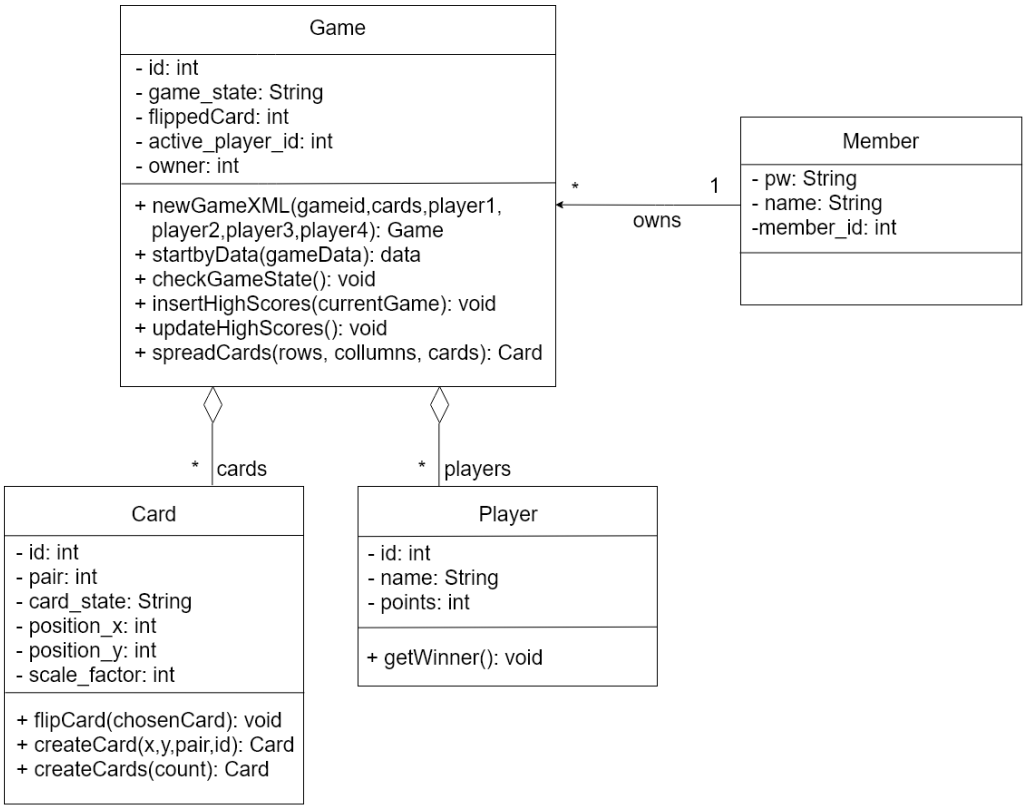
2. Struktureller Aufbau

Das Spiel ist logisch in die drei Bestandteile Model, View und Controller gegliedert. Dabei ist View die HTML-Oberfläche, welche dem Spieler im Browser zur Verfügung steht, und mit welcher er agieren kann. Der Controller beeinhaltet alle Funktionen, um den Input des Nutzers zu verarbeiten. Diese sind im Dokument *controller.xqm* gespeichert und sind mit dem Namespace "c:" versehen. Um die Anfragen zu verarbeiten, interagiert der Controller als Schnittstelle mit dem Model. Dessen Methoden sind im Dokument *methodsGame.xqm* zusammengefasst, welche den Namespace "g:" haben.

In Hinblick auf Klassen ist das Spiel wie in dem folgenden UML-Klassendiagramm aufgebaut. Ein Member mit Login-Daten und einer ID kann mehrere Spiele haben. Ein Spiel wiederum beinhaltet alle Daten und Methoden die notwendig sind für die Verwaltung des Spieles. Zudem ist die Member ID im Attribut *owner* gespeichert. Ein Spiel besteht aus mehreren Karten und gegebenfalls mehreren Spielern.

Die Karten-Klasse beinhalten alle Attribute zur Beschreibung individueller Karten sowie Funktionen zur Kartenlogik. Die Spieler Klasse beinhaltet die Id, Name und Punktzahl eines Spielers und stellt die Funktion bereit, welche ermittelt ob ein Spieler ein Gewinner ist oder nicht.

Figure 1. Klassendiagramm zum Spiel



3. Speicherung Spieldaten in Datenbanken

Für die Speicherung der Daten werden in BaseX drei Datenbanken angelegt: *XSLT_members*;, in der die Anmeldedaten der Spieler gespeichert werden, *XSLT*, welches die Informationen über die erzeugten Spiele speichert und *XSLT_highscores* in welcher die Daten zur Bestenliste zu finden sind. Diese Trennung ist keine Notwendigkeit für die gegebene Implementierung, sondern soll rein zur logischen Trennung der Daten dienen. Eine logische Trennung der Daten ist sinnvoll, speziell wenn das Programm in der Zukfunft erweitert oder modifiziert wird. Anfangs müssen die Dantebanken mittels entsprechenden XML-Dateien initialisiert werden, welche weitestgehend aus einem leeren Wurzelknoten bestehen. Wird anschließend in der Anwendung ein neues Spiel erzeugt, so wird dieses mittels der XQuery Update Facility (bereitgestellt durch BaseX) als neuer Knoten in der XSLT Datenbank eingefügt. Auf ähnliche Weise erfolgt die Speicherung der Ergebnisse in der Bestenliste. Da diese jedoch nur die besten zehn Ergebnisse speichern soll, muss geprüft werden, ob ein Ergebniss in die Liste aufgenommen werden soll und ob dabei ein bestehender Wert verdrängt wird. Die entsprechende Logik wird in der Funktion *g:insertHighsScores()*, sowie deren Hilfsfunktion *g:updateHighScores()* umgesetzt, welche sich selbst der XQUF bedient um zunächst die alten Werte aus der Datenbank zu löschen und anschließend die neue Liste zurückzuschreiben. Die Speicherung der Anmeldedaten erfolgt mit der Funktion *g:insertMember()*, welcher Benutzername und Passwort aus der Login-Seite übergeben werden. Diese prüft mit Hilfe der Funktion *g:checkMembers()*, ob ein Eintrag mit exakt diesen Daten bereits vorhanden ist. Besteht der Eintrag bereits, wird dessen spezifische ID zurückgegeben. Wenn nicht, wird diese angelegt und zusammen mit den Anmeldedaten in der Datenbank gespeichert.

4. Interne Funktionen

4.1. Erstellen neues Spiel

Wie in Kapitel 1 beschrieben, kann in der Lobby ein neues Spiel gestartet werden. Die eingegebenen Daten (Spielernamen, Kartenanzahl) werden danach zusammen mit einer Zufallszahl als ID der Funktion *newGameXML()* übergeben. In dieser wird das Spiel-Element, wie in Kapitel 2 beschrieben, aufgebaut. Die übergebene ID wird zur ID des Spiels, per Default ist der erste eingetragene Spieler der erste aktive Spieler und für jeden eingetragenen Spieler-Namen wird ein Spieler-Knoten mit dem entsprechenden Namen und einer Punktezahl von Null angelegt. Um für die gewählte Anzahl an Kartenpaaren die entsprechende Menge an Karten-Knoten zu erzeugen, wird innerhalb von *newGameXML()* die Methode *spreadCards()* aufgerufen. Bei deren Aufruf werden zunächst abhängig von der Kartenanzahl die gewünschte Menge an Karten-Knoten mit korrekter ID und Paar-Nummer mit Hilfe der Funktion *createCards()* erzeugt und *spreadCards()* zusammen mit der Spalten- und Zeilenanzahl als Input übergeben. Der Wert der Positions-Elemente ist jedoch für alle Karten zunächst noch Null. Erst beim Durchlauf der Funktion *spreadCards()* werden den einzelnen Karten per Zufalls-Permutation Zeilen- und Spalten-Nummer zugewiesen.

4.2. Anzeige Spiele und Highscores

In der Lobby sind wie in Kapitel 1 beschrieben, die gespeicherten Spiele, als auch die Highscores anzuzeigen. Diese Anzeigen werden jeweils mit den Funktionen *SavedGamesList()* und *HighScoreList()* ermöglicht. Erstere läd alle gepeicherten Spiele aus der Datenbank, die vom aktuell eingeloggten Spieler angelegt wurden, und gibt für jedes dieser Spiele einen Knoten zurück, der ID und die zusammengefassten Namen der Spieler des Spiels als Elemente enthält. In *HighScoreList()* wird ein Knoten "highscores" mit allen in der Datenbank *XSLT_highscores* enthaltenene Spielern befüllt, sortiert nach der Anzahl der Highscore-Punkte jedes enthaltenen Spielers. Diese Knoten werden dann in der *lobby.xml* ausgelesen und der Reihe nach angezeigt.

4.3. Spiellogik

Eine zentrale Rolle im Memoryspiel und somit auch der Programmierung spielt die Funktion *flipCard()*. Die einzige Handlungsmöglichkeit für die Spieler eines Memoryspiels ist es, eine Karte umzudrehen, wodurch gleichzeitig der weitere Spielverlauf beeinflusst wird.

Die Methode *flipCard()* erhält als Parameter die eindeutige ID der Karte im Spiel, ändert den Spielzustand in der Datenbank wie im Folgenden erklärt entsprechend ab und erstellt damit mithilfe des SVG-Creators die entsprechende GUI und gibt diese anschließend zurück. Da Änderungen an der Datenbank erst mit dem Beenden der Methode durchgeführt bzw. sichtbar sind, müssen entsprechende Änderungen sowohl in der Datenbank als auch in einer lokalen Kopie des Spiele-Datensatzes durchgeführt werden. Letzterer wird anschließend an den SVG-Creator übergeben.

Zunächst wird überprüft, ob die Karte noch umgedreht auf dem Spielfeld liegt (@card_state="hidden"). Trifft dies nicht zu, erfolgt keine Aktion und die Methode wird bereits wieder beendet, da die entsprechende Karte demnach entweder bereits aufgedeckt auf dem Spieltisch liegt oder zu einem bereits gefundenen Paar gehört und nicht mehr im Spiel ist.

Da zu einem Spielzug das Aufdecken von zwei Spielkarten gehört, wird nun überprüft, ob es sich hierbei um die erste oder zweite Karte des Zuges handelt. Im ersten Fall (first_card=0) wird die aktuell umgedrehte Karte als erste Karte markiert, das Spiel-Element *first_card* also auf die ID der Karte gesetzt. Außerdem wird die aktuelle Karte umgedreht (@card_state="flipped").

Wurde dagegen bereits eine Karte umgedreht, *first_card* also bereits mit einer Karten-ID belegt, wird geprüft, ob die beiden Karten ein Paar bilden, deren beiden *group*-Attribute also übereinstimmen. In diesem Fall werden dem aktuelle Spieler zwei Punkte gutgeschrieben, die Gruppen-ID im Spiel in *lastpair* gespeichert und beide Karten aus dem Spiel genommen.

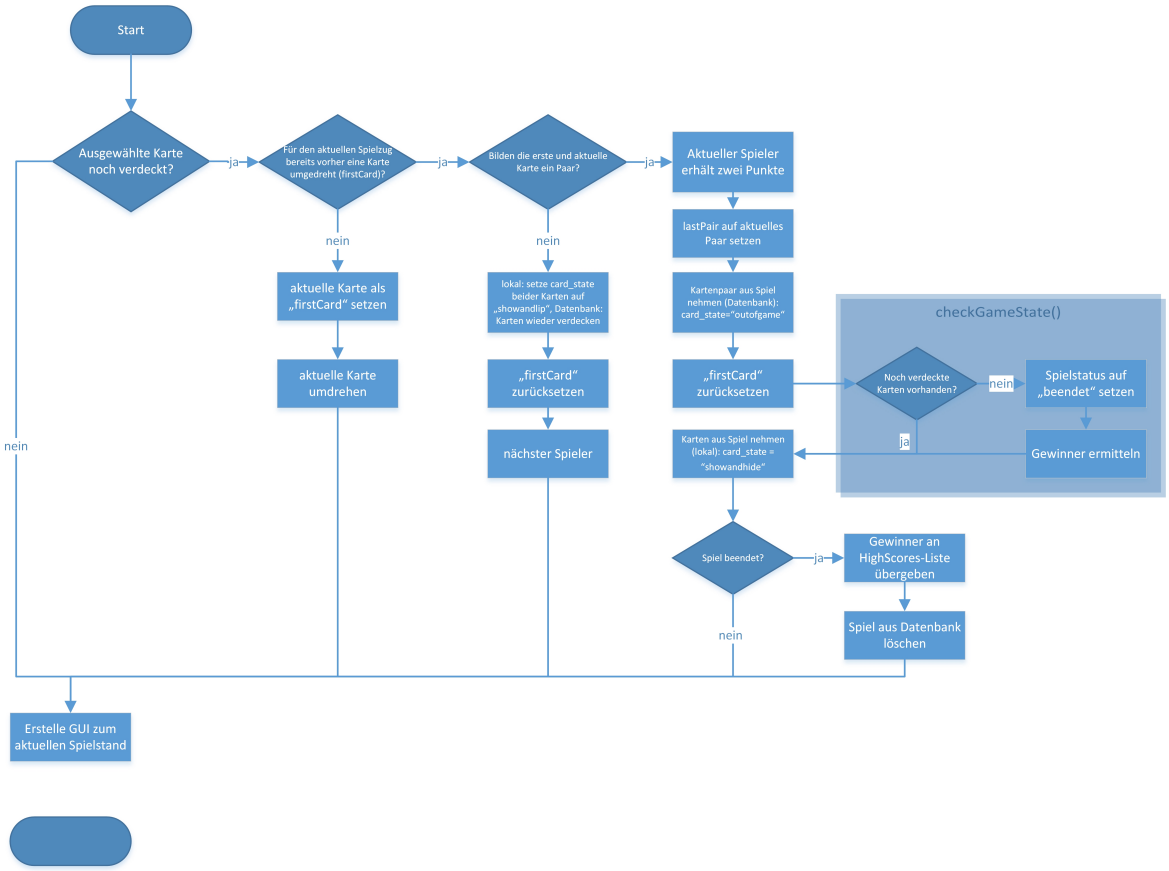
Hierfür werden ihre *card_state*-Attribute in der Datenbank auf "outofgame" gesetzt, lokal auf "showandhide", wodurch diese in der GUI zunächst aufgedeckt angezeigt und kurz darauf ausgeblendet werden. Außerdem muss an dieser Stelle überprüft werden, ob das Spiel hiermit bereits beendet ist. Dies geschieht durch die Überprüfung der Methode *checkGameState()*, ob nun bereits alle Karten umgedreht wurden bzw. keine verdeckten Karten mehr auf dem Tisch liegen. Trifft dies zu, so wird der Zustand des Spiels (*game_state*) auf "finished" gesetzt sowie die Gewinner des aktuellen Spiels ermittelt und im Spiele-Eintrag als *winners*-Element eingefügt.

Außerdem werden über die Methode *insertHighScores()* die Gewinner des Spiels bei Bedarf in die HighScores-Liste eingetragen.

Sollte es sich bei den beiden umgedrehten Karten um kein Paar handeln, ist der nächste Spieler am Zug und *active_player_id* wird entsprechend gesetzt. *firstCard* wird zurückgesetzt und die Karten werden wieder umgedreht, in der Datenbank *card_state* also auf "covered" gesetzt und lokal zwecks Anzeige des Motivs der Karten und anschließendem Umdrehen auf "showandflip".

Der eventuell nun veränderte Eintrag zum aktuellen Spiel wird nun an den SVG-Creator übergeben, welcher die neue GUI erstellt. Anschließend wird diese zurückgegeben.

Figure 2. Kontrollflussgraph zur Funktion



5. Erstellung HTML Nodes mit XSLT

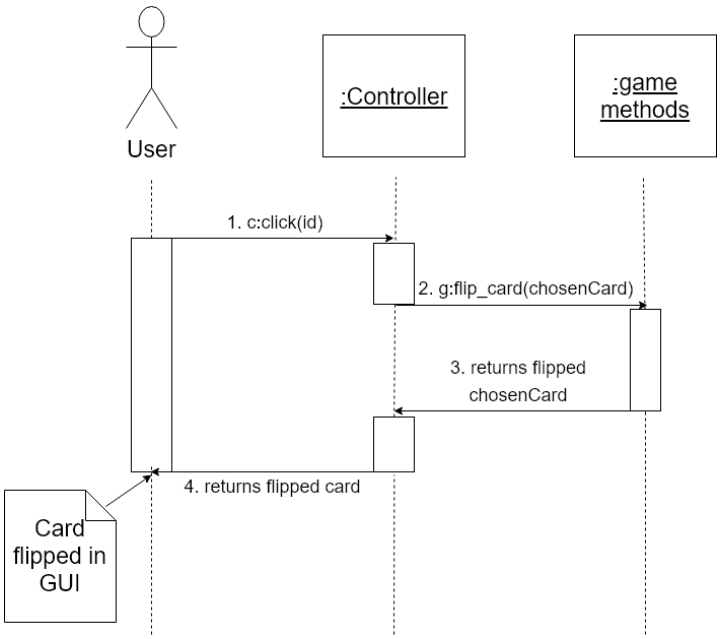
Konzeptionell lässt sich das Spiel in zwei Bereiche einteilen: Zum einen existiert die Lobby, in der die Highscore-Liste zu finden ist, neue Spiele erstellt und alte Spiele geladen werden können. Zum anderen gibt es die Darstellung für das Memory-Spiel selbst, mitsamt der Karten und dem User-Interface.

Diese Zweiteilung lässt sich in ähnlicher Weise auch in der Implementierung wiederfinden, wobei jedoch Unterschiede in der jeweiligen Umsetzung bestehen. Der statische Teil der Lobby liegt direkt als Datei(*Lobby.xml*) vor, und nutzt RestXQ um die übrigen Elemente einzufügen (Highscoreliste, Liste der gespeicherten Spiele). Auch wird die Instanz des XForms-Modell mittels RestXQ erzeugt. Für die Darstellung eines aktiven Spiels wird hingegen ein anderer Ansatz verfolgt, welcher sich auf XSL Transformationen stützt um aus den Spieldaten die aktuelle Sicht zu erzeugen. Da diese Umsetzung alle nötigen Informationen aus der Datenbank und der Session-ID bezieht, lässt sich das Laden und Speichern eines Spiels damit mit wenig zusätzlichem Aufwand realisieren. Um aus den Daten die Ausgabe mittels eines XSL Stylesheets zu erzeugen wird die Funktion *xslt:transform($input, $stylesheet)* auf die XML Daten angewendet. Diese verwendet als Default den Xalan Prozessor für die Transformationen, bzw. den Saxxon Prozessor, falls verfügbar. Mittels des hier verwendete Stylesheet(*svg_creator.xsl*) erzeugt dieser Prozessor das XHTML Dokument (inklusive XForms-Modell und Kontrollen), welches dem Nutzer anschließend durch den Browser als aktuelle Sicht angezeigt wird. Da XSLT unter anderem Schleifen unterstützt, lässt sich auf diese Weise eine eindeutige Submission für jede Karte erzeugen, aufgrund derer übermittelt wird, welche Karte vom Spieler selektiert wurde. Mittels *xsl:choose* kann eine Fallunterscheidung getroffen werden, um beispielsweise zu entscheiden, welche Seite einer Karte angezeigt werden soll. Auch die finale Zusammenfassung der Spiels, inklusive der Bekanntgabe der Gewinner erfogt mittels des Stylesheets. Die Neu-Erzeugung des Dokuments bringt für das Memory-Projekt einige Vorteile mit sich und zeigt akzeptable Performanz. Für größere Projekte sollten jedoch weitere Optimierungen vorgenommen werden oder ein anderer Ansatz verfolgt werden, um ausreichend schnell auf Nutzer-Eingaben reagieren zu können.

6. Kommunikation Client/Server

Die Requests und Responses zwischen Client und Server im Memory Game sind im folgenden Interaktionsdiagram graphisch am Beispiel der *flipcard()* Methode dargestellt. Das Beispiel zeigt exemplarisch wie der ganze Prozess generell, chronologisch abläuft. Am Anfang erfolgt die ursprüngliche Interaktion zwischen User und Client über das Eingabefenster. Dieses ist mit XForms realisiert. Die Kommunikation des erfolgten Requests zum Server geschieht dann duch den Controller, welcher die dazu erforderlichen RestXQ Funktionen beinhaltet.(In diesem Beispiel *flipcard()*) Der resultierende Methodenaufruf, also der Response, wird mit Hilfe von XQuery über URL's ausgeführt.

Figure 3. Interaktionsdiagram für flip_card() Methode



7. Reflektion

XML basiert auf der Idee, eine universelle Darstellung zur Verfügung zu stellen, mittels derer Anwendungen Daten austauschen können und dabei weiterhin durch einen einfach nachzuvollziehenden, hierarchischen Aufbau menschen-lesbar zu bleiben. Damit hat sich XML, neben vielen anderen Anwendungsbereichen, als eine Schlüsseltechnologie für das Internet erwiesen. Im Verlauf des Praktikums wurde deutlich, dass gerade durch die wenigen Regeln welche an ein XML Dokument gestellt werden eine Vielfalt von Anwedungsmöglichkeiten besteht. Da XML selbst dabei nur zur Representation von Daten dient, ohne dem Nutzer eine umfangreiche Syntax aufzuzwingen, existiert eine Vielzahl von Erweiterungen zur Manipulation und Verwaltung der Daten, welche ihrerseits eine Vielfalt an Anwedungsbreichen eröffnen. Die Umsetzung des Projekts stellte dabei durch die Vorgaben eine ganze Reihe neuer Herausforderungen bezüglich der Implementierung. So mussten viele einfache Basisfunktionen des Spiels, durch vergleichsweise komplexe RestXQ- und XQuery Funktionen nachgebildet werden. Auch das Auffinden von Fehlern im Code war weitaus komplizierter, da in vielen Fällen wenig Informationen über die Ursache bereitgestellt werden konnten. Umso wichtiger wurde ein guter Überblick über den Programmablauf und ein sehr feines iterieren des Codes. Obwohl in diesem Projekt viele der Anforderungen künstlich gegeben waren, zeigten sich damit sehr deutlich auch viele praktische Engineering-Aspekte, da auch in der Praxis die Freiheitsgrade der Implementierung durch eine Vielzahl bestehender Requirements eingeschränkt sind.

Die Organisation und Betreuung im Praktikum war sehr übersichtlich und sinnvoll gestaltet. Die verschiedenen Themengebiete wurden logisch auf die 5 Blätter aufgeteilt und die Aufgaben waren gut geeignet um die Konzepte zu verinnerlichen. Die Balance zwischen Einarbeitungsphase und Eigenarbeitsphase ermöglichte es, so wie die theoretische als auch die praktische Seite von XML zu erlernen.

Die Organisation der Arbeit im Team verlief ebenfalls reibungslos. Bei circa jede 2 Wochen stattfindenden Treffen wurden konkrete Meilensteile besprochen und gesetzt; sowie für das Projekt an sich als auch für jedes einzelne Teammitglied. Lösungen zu konkreten Aufgabenteilen wurden immer zusammen durchgesprochen und es wurde immer gegenseitiges Feedback gegeben. Die Kommunikation erfolgte über eine Whatsapp Gruppe und Slack, der Datenaustausch über GitHub.