Team Xlink Projekt
Dokumentation BlackJack

Andreas Binder
Julian Feldmeier
Christian Lang
Moritz Münker

Team Xlink Projekt Dokumentation BlackJack

by Andreas Binder , Julian Feldmeier , Christian Lang , and Moritz Münker

Table of Contents

Chapter 1. Einleitung

Blackjack gehört zu den bekanntesten und meistgespieltesten Glücksspielen in Casinos weltweit. Es entwickelte sich aus einem französischen Kartenglücksspiel aus dem 18.Jahrhundert, genannt "Vingt et un" (Übersetzt: "21"). Blackjack galt es im "XML-Technologie Praktikum" wie der Name schon voraus sagt, mit Hilfe von XML und des X-Stacks als Webanwendung umzusetzen. Obwohl XML für das Strukturieren von Daten in Textform bekannt ist, so können trotzdem Webanwendungen plattformunabhängig modelliert und implementiert werden. Sei es die Implentierung der Benutzeroberfläche mit SVG und XSLT, die Funktionalität des Spiel mit der Abfragresprache XQuery, oder BaseX als Datenbank zur Speicherung von XML-Daten und zur Ausführung von XQueryFunktionen.

Die vorliegende Dokumentation beschreibt die Planungs- und Konzeptions- sowie die Implementierungsphase des Spiels.

Chapter 2. Regeln unserer BlackJack Variante

Für Blackjack gibt es mehrere verschiedene, bekanntere und weniger bekanntere Regeln. Die Fassung unserer Spielversion lautet nun wie folgt:

Die Anzahl an 'Paketen' französischer Karten, à 52 einzelner Karten, kann vor Beginn des Spiels festgelegt werden. Als default Wert werden 5 Stacks, also 260 Karten benutzt. Es können mindestens ein Spieler und maximal 6 Spieler gegen den Dealer das Spiel spielen. Es geht für jeden Spieler darum, allein gegen die Bank zu gewinnen. Dabei hat das Spielverhalten des einen Spielers abgesehen von den gezogenen Karten aus dem Deck keinen Einfluss auf den Spielverlauf eines anderen Spielers. Zu Beginn des Spiels erhält jeder Spieler zwei und der Dealer eine Karte. Alle Karten werden offen auf dem Spieltisch gelegt. Nun ist jeder Spieler der Reihe nach dran: Ist ein Spieler am Zug, hat er drei Optionen. Er kann entweder seinen Zug beenden ("stand"), eine weitere Karte ziehen ("hit") oder seinen Einsatz verdoppeln und dann eine weitere Karte ziehen ("Double Down"). Zieht er eine weitere Karte, so wird diese offen zu seinem Blatt hinzugelegt. Das sogenannte 'splitten' bei zwei gleichwertigen Karten ist in unserer Version nicht möglich. Beendet er seinen Zug, so wartet er bis die anderen Spieler und der Dealer an der Reihe waren. Der Dealer verfolgt strikt die Strategie "Stand on soft 17, draw to 16". Dies bedeutet, dass er solange zieht bis er 17 oder mehr Punkte erreicht hat. Dabei wird ein Ass immer mit 11 Punkten gezählt, es sei denn er hätte damit mehr als 21 Punkte erreicht.

Mit der Beendigung des Zugs des Dealers wird das Spiel beendet und die Gewinner und Verlierer ermittelt. Das Ziel des Spiels ist es, so nahe wie möglich an die Punktzahl 21 heranzukommen: Zweier bis Zehnerkarten zählen entsprechend ihrer Zahl; Bube, Dame und König jeweils 10 Punkte; ein Ass zählt so lange als 11 Punkte bis der Punktestand 21 ist, kommt der Spieler darüber, zählt das Ass als 1 Punkt. Erhält ein Spieler über 21 Punkte, hat er automatisch verloren. Ist der Endpunktestand des Dealers höher als 21, haben alle Spieler mit weniger als 22 Punkten gewonnen. Ansonsten gewinnen diejenigen Spieler, die näher an 21 Punkten sind als der Dealer. Hat ein Spieler dieselbe Punktzahl wie der Dealer, ist das Spiel für ihn unentschieden.

Chapter 3. Eingesetzte Technologien

Für die Implementierung des Spieles haben wir folgende Technologien eingesetzt:

Serverseitig haben wir den Stomp-BaseX Server vom stomp-branch des offiziellen Basex Github Repositories genutzt um XML Daten zu speichern und XQuery Funktionen auszuführen. Darauf aufbauend haben wir RestXQ für das Mapping von URLs auf XQuery Funktionen eingesetzt.

Clientseitig haben wir auf XSLT, XHTML, CSS und JavaScript gesetzt. JavaScript wird hierbei ausschließlich für die WebSocket Verbindungen und Bootstrap benötigt und enthält ansonsten keine Spiellogik. Bootstrap wurde benutzt um die Lobby, den Login und das Leaderboard responsive und schön zu gestalten.

Chapter 4. Projektstruktur

Die Projektstruktur ist folgendermaßen aufgebaut:

Im Ordner "WEB-INF" befinden sich die Konfigurationsdateien für BaseX und den eingebetteten Jetty Webserver.

Im Ordner "static" befinden sich die JavaScript Dateien für den websocket und die STOMP Erweiterung.

Der Ordner “xslt” in "xlink_blackjack" enthält für jeden relevanten Screen eine eigene xsl-Datei, die das entsprechende Template darstellt und durch eine transformation mit Inhalt befüllt wird.

Im Ordner "xqm" in "xlink_blackjack" befinden sich alle xQuery Dateien. Diese sind die logischen Bausteine des Spiels und sind in verschiedenen Dateien sortiert.

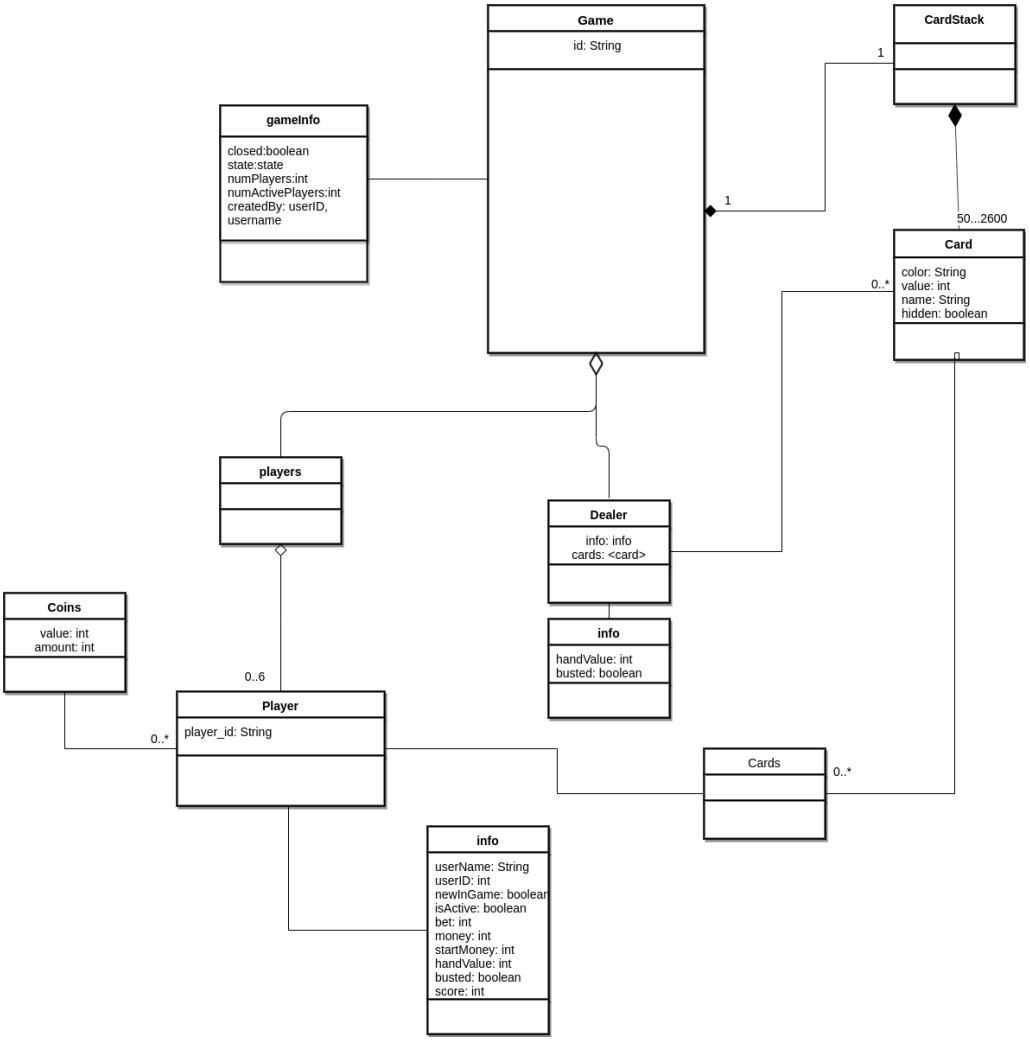
In der datei "managament.xqm" befinden sich zwei wichtige Funktionen zum erstellen der Datanbank (setup) und zum resetten der Datenbank (reset). Die weiteren Funktionen sind primär zum debuggen gedacht.

In der lobby.xqm findet man alle Funktionen rund um die Lobby, Registrieren, Löschen und Leaderboard. Zum erstellen eines Spiels kommt die Datei newGame.xqm zum Zuge, dort ist auch nochmals ersichtlich wie ein Spiel datentechnisch aufgebaut ist.

Das eigentliche Spiel findet sowohl in game.xqm als auch in gameMoves.xqm statt. Ersteres bietet Funktionen rund um das joinen, websocket-anbindung und verlassen des Spiels. In gameMoves.xqm wird die Spiellogik implementiert.

Chapter 5. Konzeption und UML Klassendiagramm

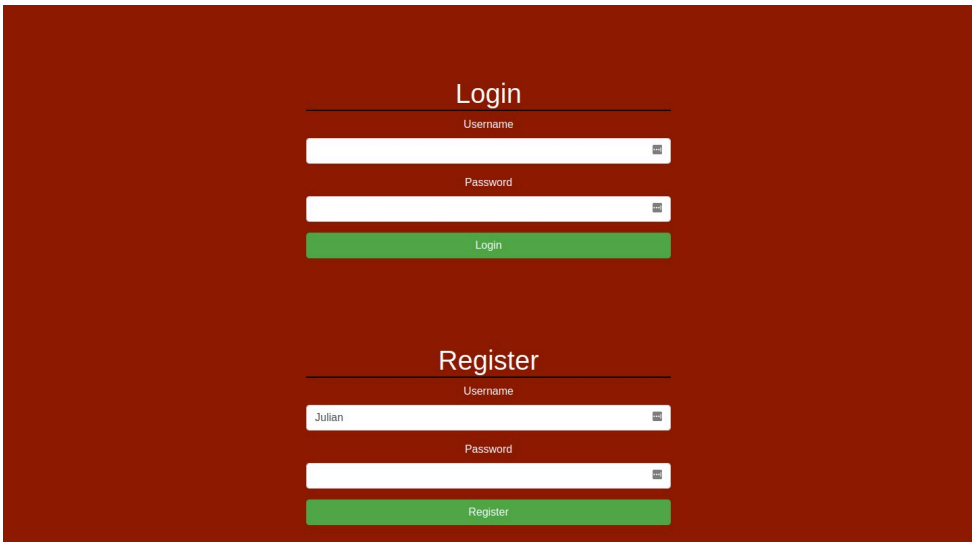
Anhand der in Kapitel 2 festgelegten Regeln wurde ein UML-Klassendiagramm erstellt, das die Basis für die Architektur des Spiels und der grafischen Benutzeroberfläche bildet. Das UML-Diagramm sieht aus wie folgt:



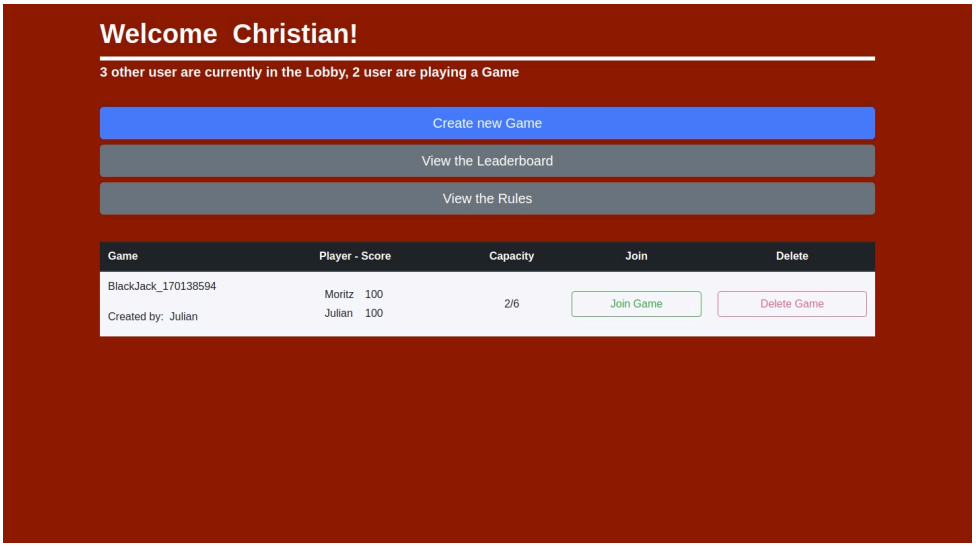
Das Klassendiagramm stellt die Komponenten eines einzigen Spiels dar. Zentrale Klasse ist die Klasse 'Game' mit dem Schlüsselattribut 'id'. Ein Game besteht aus einer Liste von Spielern (maximal 6 Spieler), einer festlegbaren Anzahl an Decks (Kartenstapel) und einem Dealer. 'players' besteht aus maximal sechs Objekten aus der Klasse 'Player', in welcher alle registrierten Spieler gelistet sind. Schlüsselattribut in dieser Klasse ist 'userID'. 'Player' steht als einzige Klasse in Beziehung zu 'Coins', diese haben einen Wert 'value' und eine Menge 'amount'. Cards werden in unserer Version generisch generiert, das heisst je nach Zahl 'value' und Farbe 'color' wird die Karte entsprechend nach den beiden Schlüsselattributen zusammengesetzt. Zudem hat jede Karte das Attribut 'isHidden', das speichert, ob die Karte auf dem Spieltisch sichtbar ist oder nicht. 'Player' und 'Dealer' stehen in Beziehung zu 'Cards', da Sie logischerweise im Spiel Karten auf der Hand haben.

Chapter 6. Beschreibung der grafischen Oberfläche

In der ersten Abbildung sieht man den Login und Registerungs-Bildschirm des Spiels. Dieser ist sehr einfach und übersichtlich gehalten. Es wird lediglich ein Username und ein Passwort verlangt und schon ist man bereit für seine erste Runde.

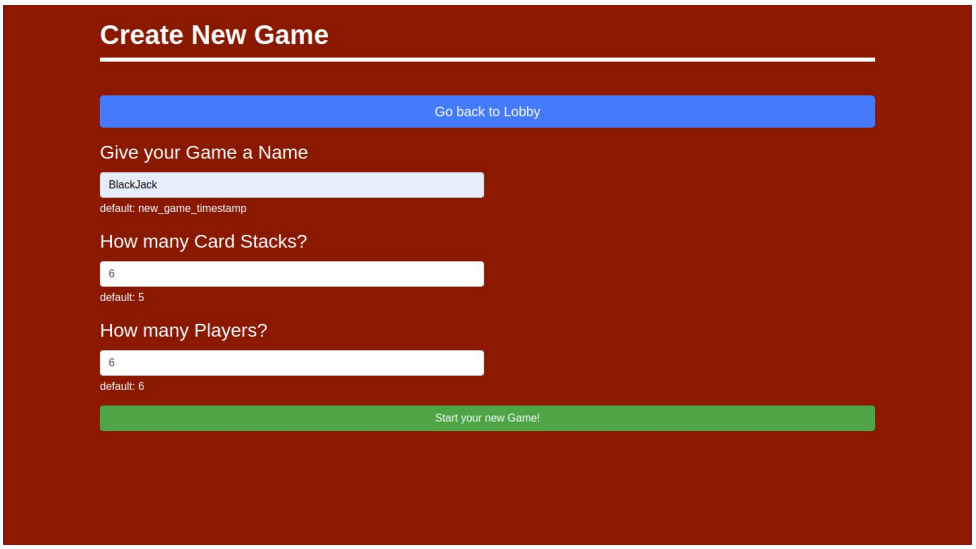


Die nächste Abbildung zeigt nun die Lobby unseres Spiels. Man hat die drei Möglichkeiten ein neues Spiel zu erstellen, das Leaderboard mit einer Highscore Liste zu betrachten oder sich noch einmal die Regeln zu Gemüte zu führen. Darunter ist noch eine Liste an aktuell stattfindenden Spielen zu finden. Man sieht wer das Spiel erstellt hat und wie die Scores im Moment aussehen. Falls noch genügend Platz ist, kann man den Spielen auch beitreten.

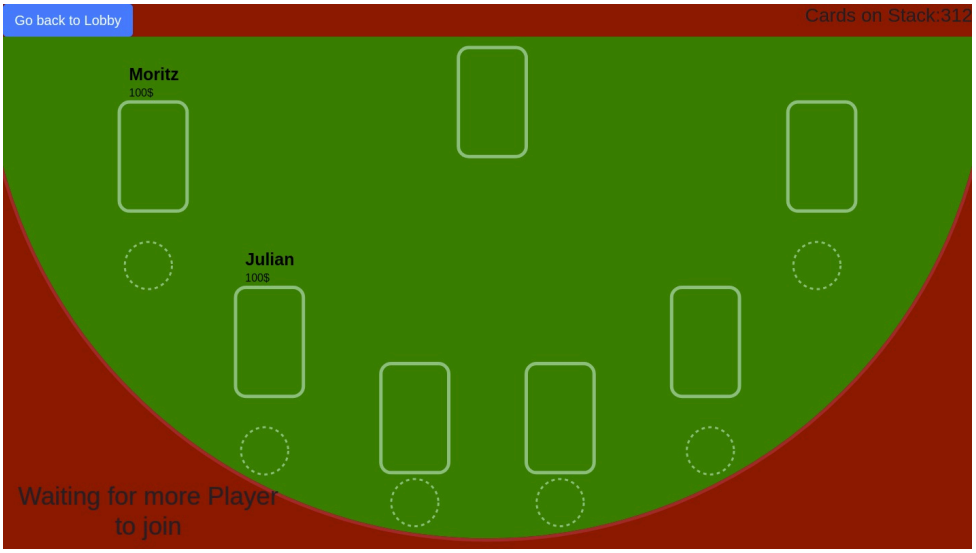


Nachfolgende Abbildung zeigt den Screen, der bei einer neuen Spielerstellung erscheint. Man kann das Spiel nach belieben benennen oder einen zufualligen Namen generieren lassen. Zusätzlich kann man eine Anzahl an Kartendecks für das Spiel festlegen. Zudem legt man noch die maximal erlaubte

Anzahl an Spielern fest. Mit einem Klick auf "Start your new Game!" kann es auch schon losgehen.

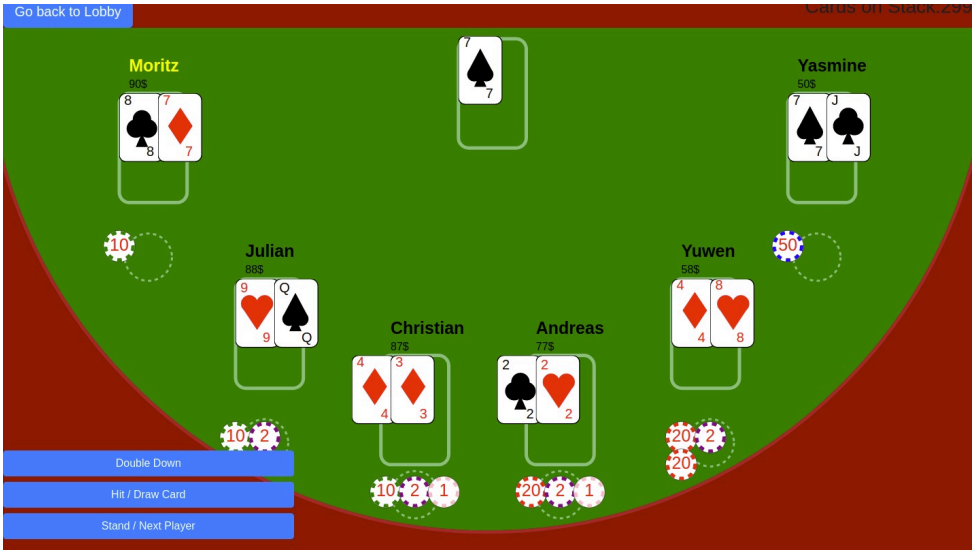


Fortfolgende Abbildung zeigt den Zustand eines neu erstellten Spiels. Der Spielersteller kann so lange warten bis alle Plätze belegt sind, um dann das Spiel zu starten. Er kann aber auch schon mit weniger Personen die erste Runde beginnen lassen. Solange noch nicht gestartet wurde steht unten links für alle sichtbar "Waiting for more Players to join".

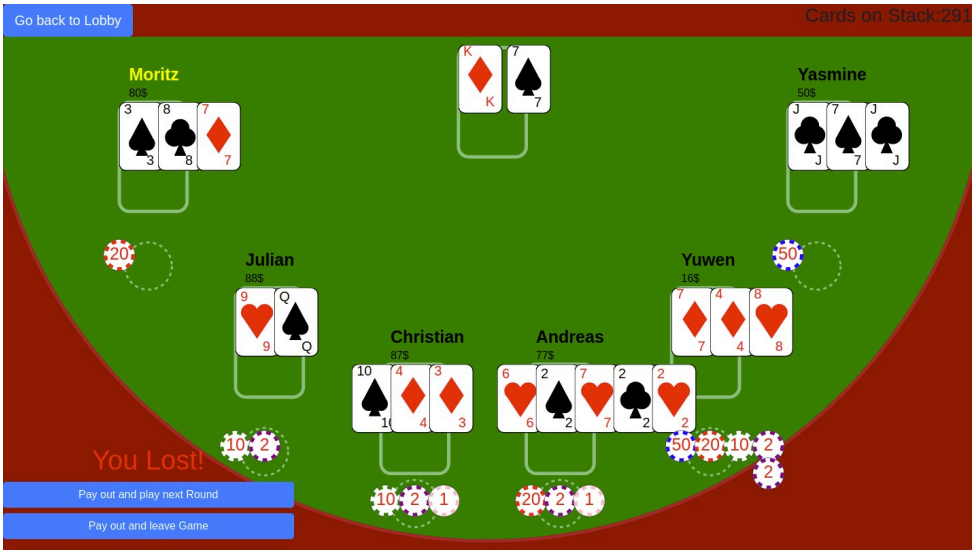


Die folgende Abbildung zeigt eine exemplarische Spielszene, nachdem die Einsätze zu Beginn abgegeben und danach jeweils Karten ausgeteilt worden sind. Die Einsätze der jeweiligen Spieler entspricht dem Wert der Chips die am jeweiligen Platz abgelegt wurden. Unter den Namen ist das Guthaben der Teilnehmer dargestellt. links unten finden sich dann drei Buttons für drei verschiedene Auswahlmöglichkeiten (vgl. "Regeln unser-

er BlackJack Variante"): "Double Down", "Hit / Draw Card" oder "Stand / Next Player".



Sobald die Runde beendet ist werden die Gewinner und Verlierer ermittelt und fortfolgend der Gewinn ausbezahlt. Anschließend kann jeder Teilnehmer entscheiden, eine weitere Runde zu spielen oder das Spiel zu verlassen. Da unsere Version auf Multi-client basiert, gibt es kein wirkliches "Ende" der Session, das Spiel läuft unendlich weiter solange noch Spieler im Spiel sind. Das endgültige 'Ende' ist das Löschen des Spiels.



Nun ein paar Sätze zur Beschreibung der Generierung der Spielkarten: Wir haben versucht die Karten so konsistent und redundanzfrei wie möglich zu halten, einzig die Symbole wie Karo oder Herz sind statisch definiert. Die Basis für eine Karte bildet "cardShape", eine leere Karte. Diese wird für jede Karte von jedem Spieler einzeln konfiguriert. Dies geschieht durch eine for each Schleife über "player", in welcher wiederum über die jeweilige "card" iteriert wird. Zunächst wird der Kartentyp abgefragt, welcher dann in der Mitte der Karte eingefügt wird. Im Anschluss wird geschaut, um welche Art von Kartentyp es sich handelt (im Sinne von Karo, Pike, etc), um die Schrift anzupassen. Sollte es entweder Herz oder Karo sein, wird Rot als Farbe für den Kartenwert festgelegt; ansonsten ist dieser Schwarz, da die Karte ja ansonsten entweder vom Typ Pike oder Kreuz sein muss. Letztendlich wird dann der Kartenwert in der ermittelten Farbe sowohl oben links als auch unten rechts angezeigt. Die Positionierung erfolgt über ein Mapping mithilfe der "game.xqm", in der die einzelnen Startpositionen festgehalten sind.

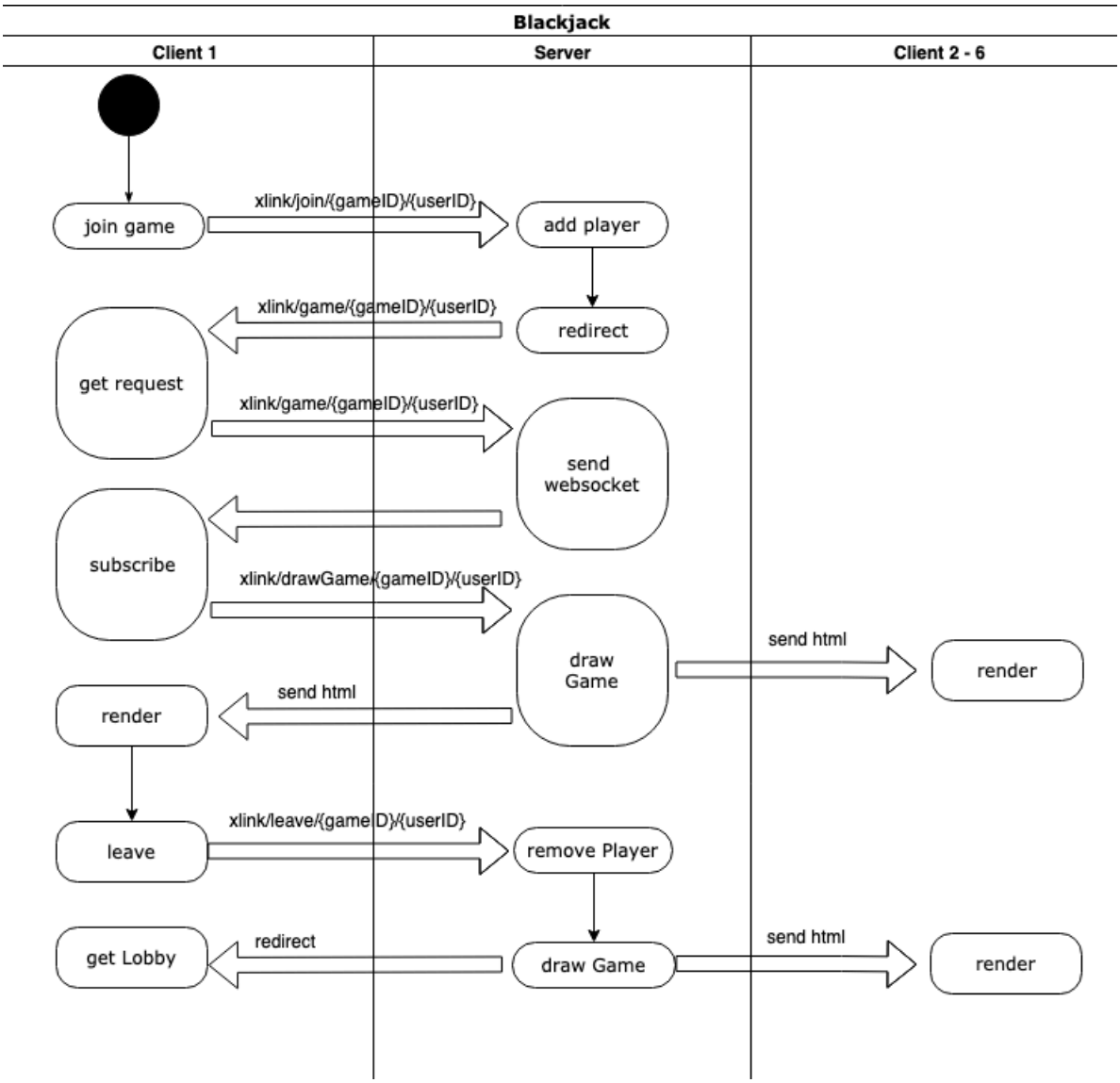
Chapter 7. Spielarchitektur

Websocket Architektur

In diesem Abschnitt wird auf die Architektur des Spiels eingegangen. Da wir eine Multi-Client Variante entwickelt haben, basiert unsere Architektur auf STOMP und Websockets. STOMP ist ein einfaches textorientiertes Nachrichtenprotokoll, das ursprünglich für Skriptsprachen wie Ruby, Python und Perl erstellt wurde, um eine Verbindung zu Unternehmensnachrichtenbrokern herzustellen. Es wurde entwickelt, um eine Untergruppe häufig verwendeter Nachrichtenmuster anzusprechen. STOMP kann über jedes zuverlässige 2-Wege-Streaming-Netzwerkprotokoll wie TCP und WebSocket verwendet werden. Obwohl STOMP ein textorientiertes Protokoll ist, kann die Nutzlast von Nachrichten entweder Text oder Binär sein. Clients können die Befehle SEND oder SUBSCRIBE verwenden, um Nachrichten zusammen mit einem "Ziel" -Header zu senden oder zu abonnieren, der beschreibt, worum es in der Nachricht geht und wer sie empfangen soll. Dies ermöglicht einen einfachen Publish-Subscribe-Mechanismus, mit dem Nachrichten über den Broker an andere verbundene Clients gesendet oder Nachrichten an den Server gesendet werden können, um die Ausführung bestimmter Arbeiten anzufordern.

Tritt ein Spieler nun einem Spiel bei ("xlink/join/{gameID}/{userID}") wird er zuerst dem Spiel hinzugefügt und wird im Anschluss auf "xlink/game/{gameID}/{userID}" weitergeleitet. Bei diesem endpoint erhält er dann ein HTML Dokument bei dem im Header die nötigen JavaScript Funktionen (stomp.js und ws-element.js) und im body ein websocket Element eingebunden wurden. Das websocket Element ist so konfiguriert, dass es zum einem einen request zu "xlink/drawGame/{gameID}/{userID}" macht und zum anderem die Subscription zu "xlink/game/{gameID}/{userID}" mit der websocket-url "ws://{host:port}/ws/xlink/game" aufbaut. Über diesen Channel kann der Server nun bei jeder Spieländerung die neue Version schicken. Die drawGame Funktion ist so aufgebaut, dass bei jedem neuem Aufruf alle verbundenen Clients über den entsprechenden Channel die neue Version geschickt bekommen. Als Channel wird die gameID benutzt, das heisst, nur Clients die eine Subscription zur gameID besitzen werden das Update erhalten. Sollte der Spieler nun eine Aktion ausführen, wird er nachdem die Änderungen serverseitig bearbeitet wurden wieder auf "xlink/game/{gameID}/{userID}" weitergeleitet, was dazu führt, dass er zum einem wieder ein websocket element bekommt und somit auch wieder die Funktion drawGame aufruft wodurch die anderen Spieler die Aktion dann auch sehen können.

Bei der Lobby funktioniert es analog. Ein User der die Lobby aufruft bekommt zuerst das websocket, erstellt eine Subscription zu "xlink/lobby/{userID}" und löst dann die drawLobby Funktion aus. Da ein User nach dem erstellen eines Spiels einfach zur Lobby weitergeleitet wird und somit ein update aller clients auslöst, können alle Spieler direkt sehen, dass es ein neues Spiel gibt.



Benutzung der RestXQ Schnittstelle

Im Folgenden werden alle relevanten Rest-Endpoints mit ihren Optionen und Erklärungen aufgelistet. Da wir sowohl mit Rest-Enpunkten als auch Websocket arbeiten kann man nicht von einer wirklichen RESTful Application sprechen, es werden in einigen Teilen allerdings doch die Grundsätze dazu eingehalten. Anzumerken an dieser Stelle ist, dass sowohl im Frontend als auch im Backend darauf geachtet wird, dass kein falscher Spielzug gemacht werden kann. Zusätzlich werden sowohl im Front- als auch im Backend alle Form-Parameter validiert.

/xlink - Seite zum registrieren und einloggen.

/xlink/setup - Erstellen der Datenbank.

/xlink/lobby/{$userID} - Aufrufen der lobby. Die userID wird in der URL mitgetragen um sie bei weiteren Anfragen zu benutzen.

/xlink/lobby/leaderboard/{$userID} - Anzeigen des Leaderboards

/xlink/lobby/docbook/{$userID} - Anzeigen des DocBooks

/xlink/join/{gameID}/{userID} - Eingebettet in den "Join" Button in der Spielliste. Mit dieser URL kann Spieler mit der entsprechenden userID einem Spiel beitreten.

/xlink/leave/{$gameID}/{$userID} - URL um das Spiel zu verlassen.

/xlink/game/{$gameID}/{$userID} - Endpoint um eine leere Seite nur mit dem entsprechend konfigurierten Websocket Element zu erhalten.

/xlink/drawGame/{$gameID}/{$userID} - Wird vom websocket aufgerufen um das erneute rendern des Spiels für alle verbundenen Clients im Channel zu erzwingen

/xlink/game/{$gameID}/{$userID}/start - Der Spielersteller kann das Spiel starten wenn genug Spieler vorhanden sind.

/xlink/game/{$gameID}/{$userID}/placeBet - Die Spieleinsätze werden als Form-Parameter übergeben: bet_amount.

/xlink/game/{$gameID}/{$userID}/draw2Cards - Dem Spieler mit id = userID werden 2 zufällige Karten vom Kartenstapel gegeben.

/xlink/game/{$gameID}/{$userID}/turn/{$turn} - Endpoint zum auslösen eines Spielzugs. Im Parameter $turn wird der entsprechende Spielzug übermittelt: 'stand', 'hit', 'doubleDown'. Ein Switch in der gameMove Funktion ruft dann je nach Parameter die entsprechende Funktion auf.

/xlink/game/{$gameID}/{$userID}/nextRound - Wenn die Runde fertig ist kann ein Spieler damit symbolisieren, dass er bereit für die nächste Runde ist. Sein Gewinn wird ausgezahlt, der Highscore angepasst und das Spielerobjekt für die nächste Runde resettet.

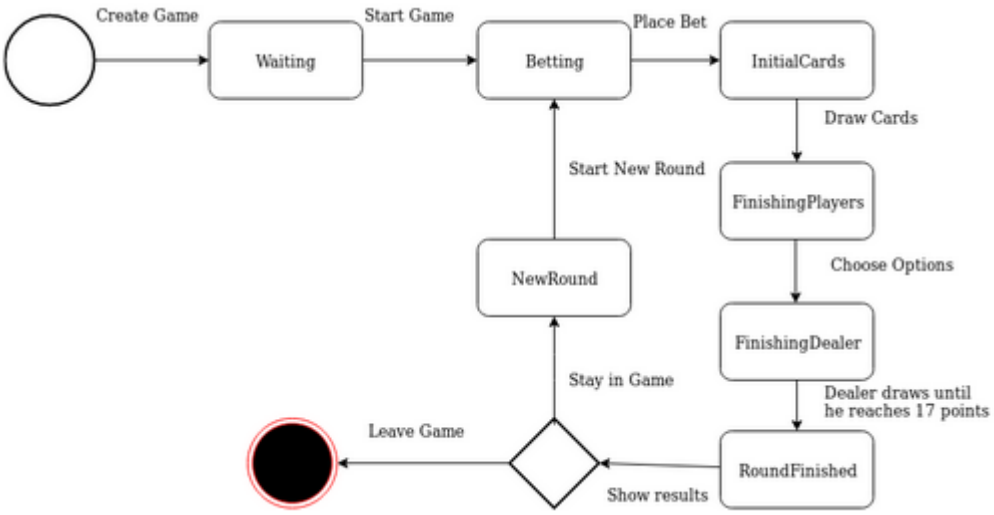
XQuery Update Facility

In so gut wie allen Funktionen werden Daten verarbeitet. Dies folgt bei uns immer einem Schema: Zuerst wird das Objekt aus der Datenbank geholt. Anschliessend wird mithilfe der non-updating expression "copy / modify / return" das Objekt wie gewünscht angepasst. Je nachdem wie komplex die Vorgänge sind, kann dies auch in mehrere Teil-Stücke aufgeteilt werden. Schlussendlich wird dann mithilfe der return(update:output()) Funktion das veränderte Objekt wieder in die Datenbank geschrieben und zum anderem dem User eine Rückmeldung gegeben.

Hierbei gibt es zwei Möglichkeiten: Meistens wird bei uns der User redirected, das heisst das Objekt wird in die Datenbank geschrieben und der User auf die Spiel oder Lobby Seite weitergeleitet. Sobald der Spieler auf der neuen Seite ankommt, ist das neue Objekt schon in der Datenbank vorhanden und der User bekommt somit die aktuellsten Daten. Die andere Variante (siehe game:leave() in game.xqm) verändert das Game-Objekt, benutzt das veränderte Objekt in der entsprechenden XSL Transformation, schickt das XSL an die Clients und updated parallel dazu das Objekt in der Datenbank.

Zustände des Spiels

Unsere Version basiert auf verschiedenen möglichen States (Zustände), welche im Spiel gerade vorherrschen können. Sobald ein Spieler ein Spiel erstellt befindet er sich im state "waiting", solange er auf weitere Spieler wartet, die seinem Spiel beitreten. Starte der Spielersteller nun das Spiel, so ist der gamestate "betting", da alle Spieler der Reihenfolge nach ihre Einsätze abgeben. Ist dieser Vorgang abgeschlossen, so wird in den state "initialCards" gewechselt und alle Spieler erhalten ihre ersten beiden Karten, sowie der Dealer seine Karte. Der state "finishingPlayer" ist die Phase des Spiels, in der der Teilnehmer die Auswahlmöglichkeiten zwischen "Stand", "hit" oder "DoubleDown" hat. "FinishingDealer" wird nicht in der Datenbank eingetragen, der state wird aber erreicht sobald der Dealer 17 oder mehr Punkt auf der Hand hat. Danach wird in den State "Roundfinished" übergegangen, in dem die Gewinner und Verlierer ermittelt werden. Bei Start einer neuen Runde, springt der State auf "NewRound" und folglich wieder auf "Betting" und der Kreislauf startet wieder von neu.


Create Game
Waiting
Start Game
Betting
Place Bet
InitialCards
Draw Cards
FinishingPlayers
Choose Options
FinishingDealer
Dealer draws until
he reaches 17 points
RoundFinished
Show results
Leave Game
Stay in Game
NewRound
Start New Round

Chapter 8. Spiellogik

Die Logik des Spiels wurde mithilfe von XQuery-Funktionen entwickelt. Anhand des Spielablaufs wir fortfolgend auf die spezifischen Funktionen der jeweiligen XQuery Dateien eingegangen.

Der Spieler muss sich vor seinem ersten Spiel ein Nutzerkonto anlegen. Dies geschieht durch die Registrierung, die Funktion hierfür befindet sich in der "lobby.xqm" Datei und lautet "register". Hierbei müssen vom Spieler username und password übergeben werden. Diese werden in der Datenbank gespeichert und der Spieler erhält zusätzlich noch eine userID.

Sobald der Spieler registriert und angemeldet ist, befindet er sich in der Lobby. Die Möglichkeiten hier werden auch über die "lobby.xqm" Datei gesteuert. Zum einen kann der Nutzer die Leaderboards aufrufen. Dabei ist die Funktion "getLeaderBoard" im Hintergrund tätig. Falls der User irgendwann ein selbst ertselltes Spiel löschen wollen würde, dann läuft dies über die "deleteGame" Methode ab. Hier ist anzumerken, dass nur der User der das Spiel erstellt hat dieses auch löschen kann.

Soll ein neues Spiel erstellt werden, so geschieht dies über die "createNewGame" Funktion in "newGame.xqm".Dabei wird die userID des Spielerstellers, der Name des Spiels, die Anzahl an gewünschten Kartendecks, und die Anzahl an Spieler übergeben. Anhand der gameID wird dann für das neue Spiel durch die "getNewGameTemplate" ein neues Spielfeld erstellt. Durch "createID" erhält das Spiel auch eine eigene Identifikationsnummer.

Will der Spieler einem bereits bestehenden Spiel beitreten, so hat er in der Lobby die Übersicht dafür. Joint er nun einem Spiel, so wird er durch die "joinGame" Methode in die Datenbank des Spiels eingetragen und wird dann auf "getGame" redirected. Dort erhält er den Websocket inklusive Subscription zu dem entsprechendem Spiel und das Spiel wird anschliessend mit "drawGame" gerendert.

Ist der Spieler im Spiel, kann es auch schon losgehen sobald eine neue Runde gestartet wird. Alle Spielmöglichkeiten werden in "gameMoves.xqm" gesteuert. Zu Beginn der neuen Runde können alle Spieler über die Methode "placeBet" nach der Reihe einen Einsatz festlegen. Die Funktion "nextPlayerNextState" stellt sicher, dass nach den individuellen Spielzügen zum nächsten Spieler gewechselt wird. Zusätzlich wird bei jedem Spielzug gecheckt ob der Spieler dazu berechtigt ist und ob der passende State gerade dafür herrscht (vgl. Spielarchitektur). Haben alle Spieler einen Betrag gesetzt so kann wiederum jeder Spieler der Reihenfolge nach zwei Karten ziehen: "draw2Cards". Als nächstes stehen den Spielern drei Optionen zur Auswahl: "doubleDown", "hit" oder "stand". Ersteres bewirkt die Verdopplung des Einsatzes ohne erneutem ziehen einer Karte. "hit" gibt dem Spieler eine weitere Karte und "stand" bewirkt nichts außer, dass nun der nächste Spieler an der Reihe ist. Es gibt solange die Möglichkeit in jeder Iteration zwischen den 3 Optionen zu wählen bis der Dealer einen Kartenwert von 17 oder höher hat. "finishDealer" wird aufgerufen und es ist den Spielern nicht mehr möglich neue Karten zu ziehen. Das Spiel ist dann zu Ende und es werden Gewinner und Verlierer ermittelt. Dies geschieht über die Methode "determineWinners", welche wiederum auf "bestWin" und "allWin" zugreift.

Zunächst zu "determineWinners", hier wird geschaut, ob sich der Dealer überkauft hat, dass heißt alle Spieler die nicht selbst gebusted sind ("busted = 'false'") erhalten eine Gewinnausschüttung gemäß ihres Einsatzes. Ansonsten wird mit der Spiel spezifischen Berechnung des Gewinners/ der Gewinner fortgefahren. Wir haben dabei folgende Notation gewählt: der result tag gibt das jeweilige Ergebnis an; der Integer 0 steht für verloren, 1 gibt ein Unentschieden/Tie an und Spieler mit einer 2 im result tag haben gewonnen. Auch hier wird nach dem "busted" Attribut gefilert. Spieler, die sich überkauft haben, bekommen eine 0 als Wert. Sollten sie dies allerdings nicht getan haben, findet ein Vergleich ihrer Handvalue mit dem Dealer statt. Sind beide Werte gleich, liegt ein Unentschieden vor (Ergebnis: 1), gilt Handvalue des Spielers größer als die des Dealers, dann Ergebnis: 2, im umgekehrten Fall wird dem Spieler eine 0 eingetragen. Anschließend erhalten Gewinner ihren Gewinn ausgzahlt, "payOutPlayer". Spieler können dann entscheiden ob sie an der nächsten großen Runde teilnehmen wollen - "nextRound" - oder das Spiel verlassen wollen, "leave". Sollte der Spieler sich entschliessen das Spiel zu verlassen werden zum einem seine Leaderboard Einträge geändert (Score Neuberechnung und Inkrementierung von 'gamesPlayed') und das Spieler Objekt wird vom Spiel entfernt. Das Verlassen des Spiels läuft dabei durch unsere Implementierung wie folgt ab: Die "leave" Funktion wird

mit den Parametern "gameID" sowie "playerID" aufgerufen; das game steht offensichtlich für das zu verlassende Spiel, der player ist der Spieler, der das Spiel verlassen möchte, identifiziert jeweils über den Primärschlüssel, die ID. Die "leave" Funktion updated zunächst die durch das Teilnehmen an dem Spiel sich ändernden Daten. Darunter fällt beispielsweise die Anzahl an gespielten Spielen als auch der gewichtete Durchschnitt des Scores. Ein neuer Highscore wird allerdings nur hinzugefügt, falls der zuletzt erreichte Punktestand den bisherigen Highscore übertrifft. Zusätzlich muss sich zum Zeitpunkt des Verlassen des Spiels sich dieses im State "roundFinished" befinden.

Sofern ein Spieler das Spiels verlässt, gilt es natürlich, die einzelnen Spezialfälle abzudecken. Einer davon ist, sollte der leavende Player der gerade aktive sein (zu erkennen an dem "isactive = 'true'" Attribut, und einer farblichen Markierung in der graphischen Oberfläche des Spiels. Im Speziellen wird in diesem Fall die "nextPlayerNextState" Funktion getriggert, welche im Anschluss beschrieben wird. Im Allgemeinen wird der Spieler, der das Spiel verlässt von der Spielerliste des Spiels entfernt, hinzu kommt einen Reindexierung der verbleibenden Spieler. Sollte der letzte verbleibende Spieler das Spiel verlassen, wird das Spiel in den State "betting" versetzt. Aus unseren State Chart Diagramm kann man dabei sehr schön erkennen, dass dadurch ein neuer Spielzyklus möglich wird und ein betretender Spieler direkt weiter spielen kann. Auf Browserebende wird der leavende Player zur Lobby weitergeleitet und an die verbleibenden Spieler wird die aktualisierte HTML Version geschickt.

Wie angesprochen, hier noch ein grober Abriss wie die "nextPlayerNextState" Funktion wirkt: die wesentliche Aufgabe besteht darin, den nächsten aktiven Spieler zu bestimmen und den vorherigen auf inaktiv ("isActive = 'false'") zu setzen. Dies passiert pro State für jeden Spieler einmal, bedeutet jeder darf einmal seine Wette abgeben, seine Karten ziehen, seinen Zusatzoptionen abwägen etc. In der Implementiertung erfolgt dies, indem über das "playerID" Attribut iteriert wird, bis eben der Spieler mit der höchsten ID (maximal 6) erreicht wird. In diesem Fall wird wieder mit 1 begonnen. Während dem laufenden Spiel begetretende Spieler werden dabei als neu im Spiel behandelt und übersprungen, wobei der Spieler als inaktiv deklariert wird und die "nextPlayerNextState" Funktion aufgerufen wird, diesmal mit dem nächsten Spieler als Argument. Beispiel: Es befinden sich 5 Spieler im Spiel. Ein neuer Spieler tritt dem laufenden Spiel bei, er erhält dabei die Spieler ID 6. Wenn Spieler 5 seinen Zug beendet hat, erkennt dass Programm, dass Spieler 6 neu dabei ist (Element newInGame = true) und überspringt diesen. Da die maximale Anzahl an Spielern dabei erreicht wird, ist Spieler 1 folgerichtig der nächste Spieler. Eine Vereinfachung, die wir hierbei gewählt haben, tritt auf, sofern sich nur ein Spieler im Spiel befindet; dabei ist kein Wechsel des "isactive" Attribut nötig, da alle Spielzüge nur von einem Spieler getätigt werden.

Chapter 9. Installationsanleitung

Wenn Sie unser Spiel austesten wollen, dann müssen Sie zuerst die "basex-stomp"-Version von dieser Internetseite herunterladen: https://github.com/BaseXdb/basex/tree/stomp. Danach muss der Zip-Ordner noch entpackt werden.

Fortfolgend navigieren Sie in diesem Ordner zu "basex-api/src/main" . Dann platzieren Sie den "xlink_blackjack" und "static" Ordner in "webapp".

Anschließend navigieren Sie wieder zurück zum Wurzelordner "basex-stomp" und führen dort den Befehl "mvn install -DskipTests" aus. Sobald dies erledigt ist, gehen Sie wieder zurück in den "basex-api" Ordner und führen dort ""mvn jetty:run" aus.

Öffnen Sie in Ihrem Browser ""localhost:8984/xlink/setup", um die basex Datenbank zu installieren. Suchen Sie danach ""localhost:8984/xlink", um auf die Login- und Register Seite zu gelangen. Danach kann es auch schon los gehen. Viel Spaß!

Chapter 10. Praktikumsreflexion

In diesem Abschnitt der Dokumentation geht es um eine Beschreibung der Organisation der Arbeit im Team sowie eine allgemeine Reflektion zur Thematik, Organisation und Betreuung im Praktikum.

Das Team besteht aus den 4 Mitgliedern Andreas Binder, Julian Feldmeier, Christian Lang und Moritz Münker. Die Projektaufgabe wurde in die vier Bereiche Architektur, Design, Logik sowie Erstellung des DocBook aufgeteilt. Dabei haben sich Moritz Münker und Andreas Binder um die Entwicklung und Gestaltung der Architektur und der Spiellogik gekümmert. Alle Mitglieder waren bei der Entwicklung des Designs tätig. Christian Lang war für die Erstellung des DocBook und Julian Feldmeier für die Ausarbeitung der UML-Klassendiagramme verantwortlich

Die Kommunikation erfolgte über persönliche Treffen und bei akutem Kommunikationsbedarf über die Plattform Whatsapp. Zwischendurch war es teilweise schwierig persönliche Treffen mit einem voll anwesenden Team zu ermöglichen, da einige Personen auch als Werkstudenten tätig sind und in den Semesterferien Vollzeit gearbeitet haben. Whatsapp hat den Vorteil, dass sie den Informationsbedarf in relativ kurzer Zeit deckt und Probleme so sehr schnell gelöst werden können. Nachteil der verwendeten Kommunikationsplattform waren eine fehlende Übersicht der zu besprechenden Themen und die Verzögerungszeit zwischen Fragen und Antworten, da logischerweise nicht jeder Vollzeit am Handy ist. Insgesamt herrschte während der gesamten Projektphase durchweg eine höchst angenehme Atmosphäre im Team, sodass wir uns immer schnell auf eine Lösung einigen konnten und uns gegenseitig unterstützt haben. Zum Ende hin musste der Termin für die Präsentation noch verschoben werden, da man mehr als möglich auf die letzten Tage vor Abgabe geschoben hat, was dann leider nach hinten los ging. Die Mitglieder haben aber aus diesem Vorfall gelernt und wollen Projekte in Zukunft anders planen.

Mit der Thematik des Praktikums ist nur einer der vier Mitglieder in unserem Team - Moritz Münker - vor Beginn des Praktikums näher in Berührung gekommen. Auch sind sonst im Studienplan sowohl im Studiengang Informatik als auch Wirtschaftsinformatik keine Pflichtmodule vorgesehen, bei denen man die Themenbereiche Document Engineering und XML Technologien kennenlernt, weshalb dieses Praktikum für jeden von uns eine große Bereicherung zum Studium ist. Die wöchentlichen Übungsblättern, haben uns sehr geholfen in das Thema XML einzusteigen und die einzelnen Komponenten am Ende zu einem fertigen Spiel mit Dokumentation zusammenzusetzen. Auch, dass mitunter sehr viel eigenständiges Arbeiten und Koordinieren innerhalb der Projektgruppe gefordert wird, ist von großem Vorteil im Hinblick auf das Berufsleben. Etwas schade ist, dass nur wenige Materialen zum Erlernen von XML-Technologien im Internet vorhanden sind. Durch anwendungnahe Beispiele in Moodle und den Vorlesungsfolien sowie durch Betreuung während der Präsenzstunden und via E-Mail, wurden Fragen, Anliegen und Unklarheiten immer rasch beantwortet bzw. gelöst. Dies alles hat dazu beigetragen, dass wir nun alle glauben, die Lernziele des Praktikums erreicht zu haben.