

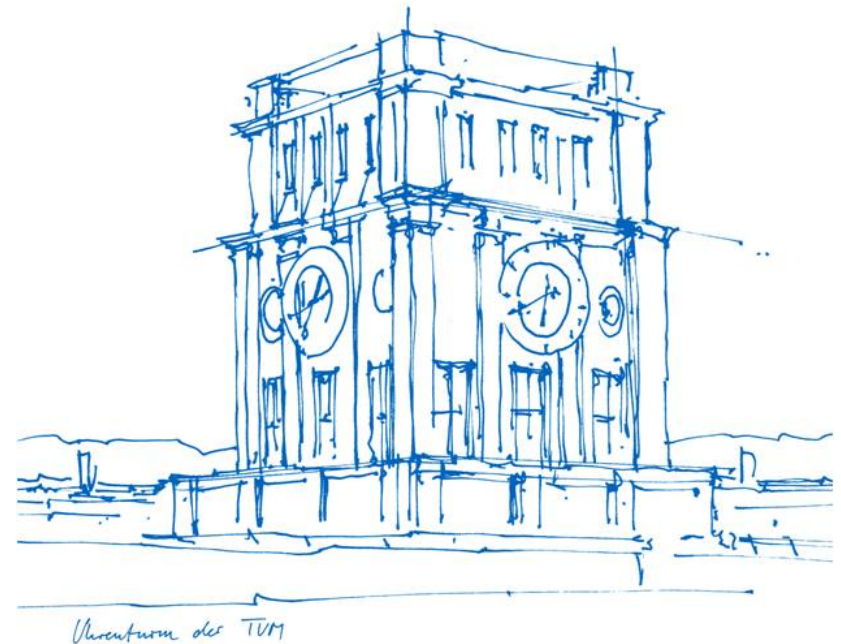
# Multi-client Black Jack mit XML Technologien

Janik Schnellenbach, Felix Hennerkes, Maximilian Karpfinger, Daniel Meint

Technische Universität München

Fakultät für Informatik

Garching, 16. August 2019

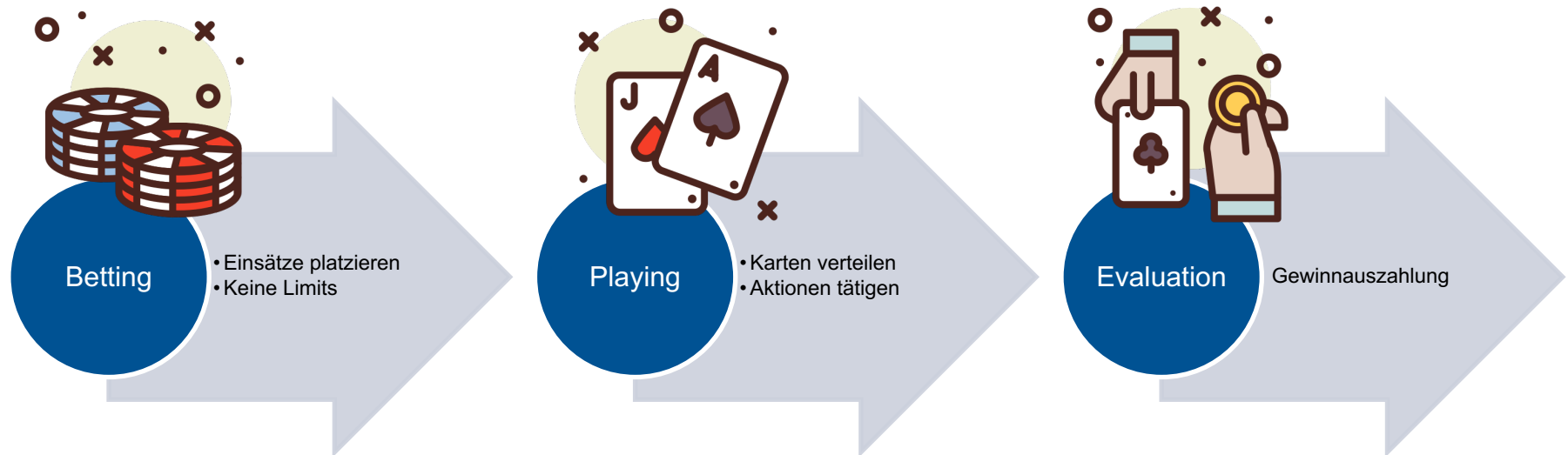


# Gliederung

1. Das Spiel
2. Architektur
3. Model
4. View
5. Controller
6. Methodik
7. Demo

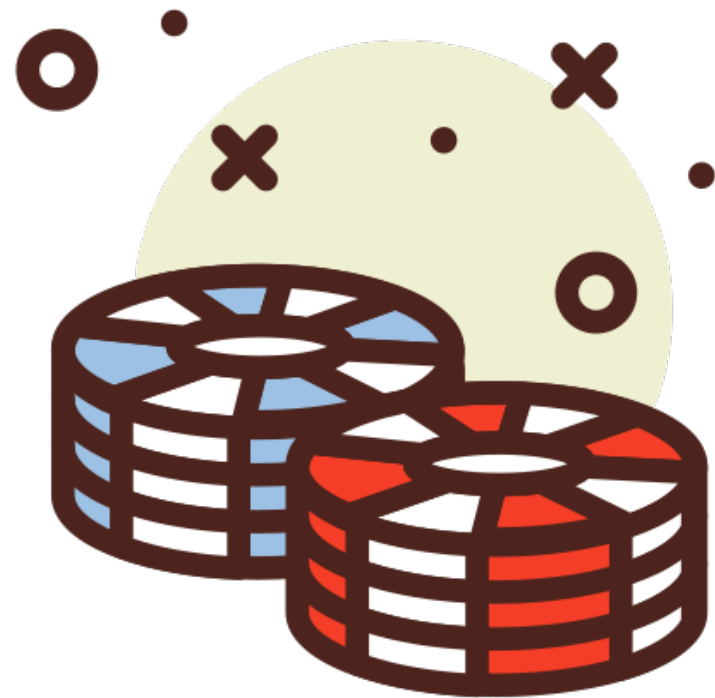
# 1. Das Spiel

# Rundenverlauf



# „Betting“-Phase

- Alle Spieler platzieren ihre Wetteinsätze
- Mindesteinsatz: \$1
- Höchsteinsatz: Guthaben des Spielers
- Automatischer Übergang in die „Playing“-Phase sobald der letzte Teilnehmer seinen Einsatz bestätigt



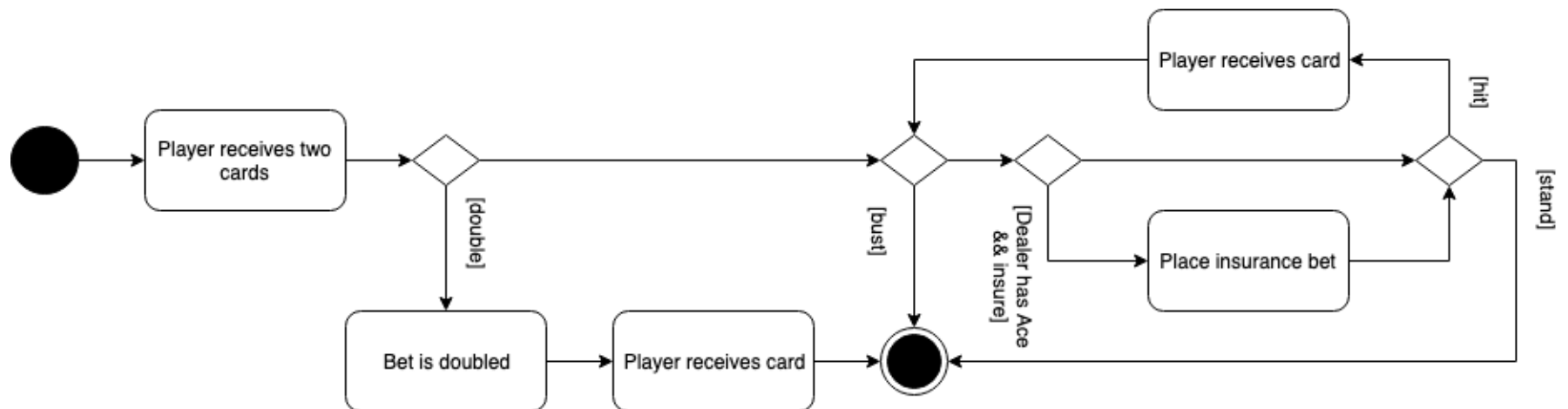
# „Playing“-Phase

- Jeder Spieler erhält zwei Karten
- Nacheinander verlangen Spieler zusätzliche Karten um Punktwert zu maximieren ohne 21 zu überschreiten
- Verschiedene Aktionsmöglichkeiten
  - *Stand*
  - *Hit*
  - *Double*
  - *Insurance*



# „Playing“-Phase

Entscheidungsmöglichkeiten eines einzelnen Spielers als Kontrollfluss



# „Evaluation“-Phase

- “*Dealer stands on soft 17*”
- Spieler gewinnt falls
  - Mehr Punkte als der Dealer ohne zu überkaufen (*bust*)
  - Beliebiger Wert  $\leq 21$  während Dealer sich überkauft
  - Sonderfall Black Jack
- Unentschieden führt zu *push*
- Andernfalls verliert der Spieler seinen Einsatz





## 2. Architektur

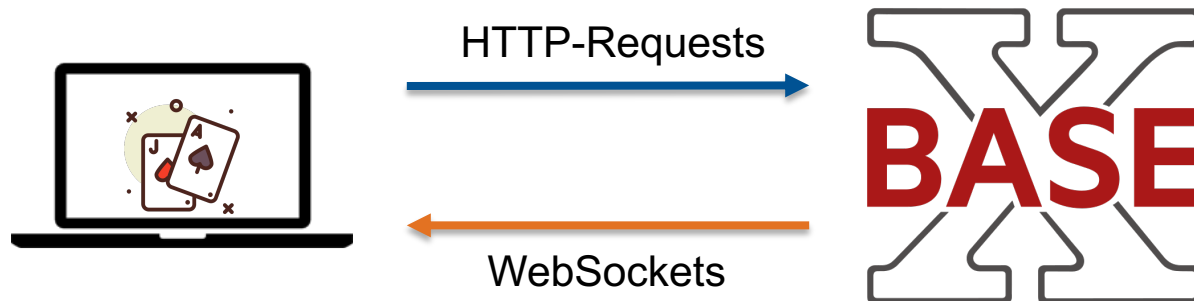
# Client-Server-Modell

- Web Anwendung
- Client: Browser-basiert
- Server: XML-Database, XQuery Prozessor, XSLT Prozessor



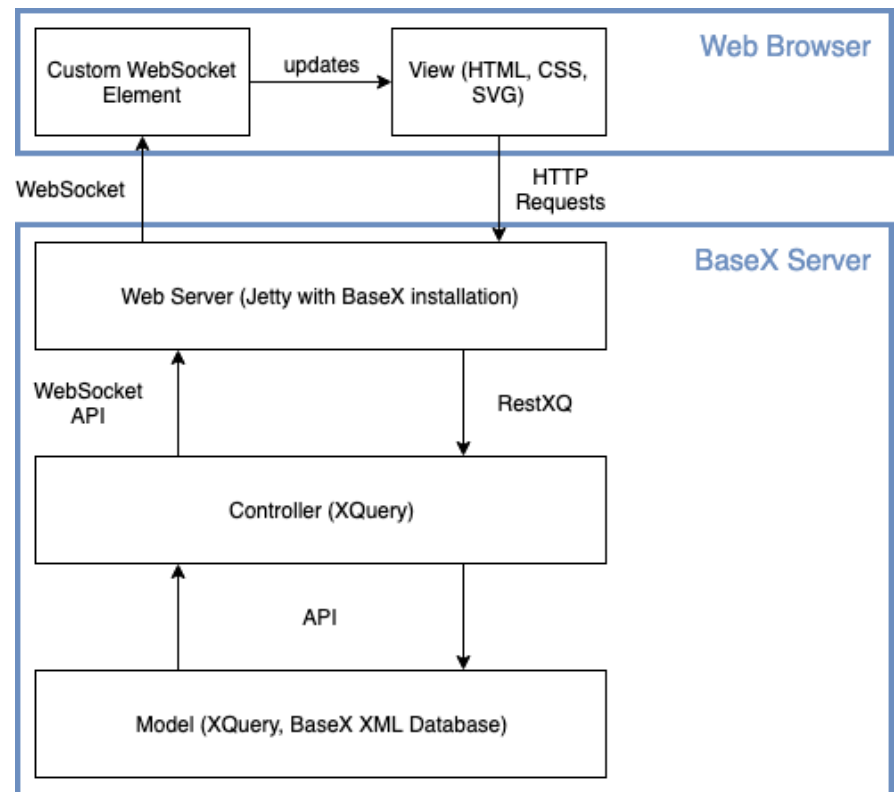
# Bidirektionale Kommunikation

- Short Polling, Long Polling, HTTP Streaming → Echtzeit ?
- WebSockets



# Model-View-Controller

- Separation of Concerns
- Push Variante

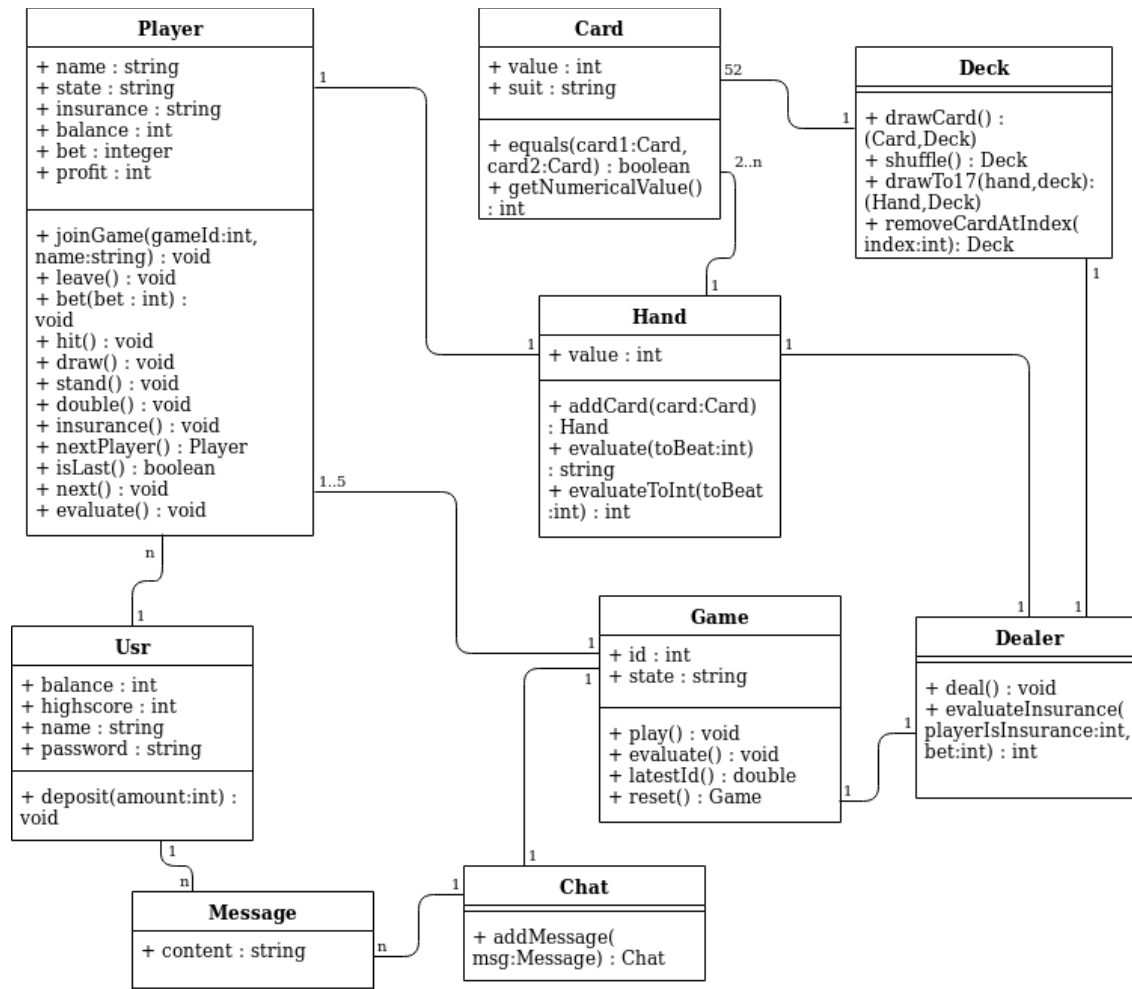


# 3. Model

# Komponente Model

- Aufgabe: Daten speichern und updaten
- Objekt-orientierter Ansatz
- Viele kleine Klassen anstatt wenig große
- Strukturiertheit, Lesbarkeit, Redundanzfreiheit, Erweiterbarkeit

# UML Klassendiagramm



# Game Class

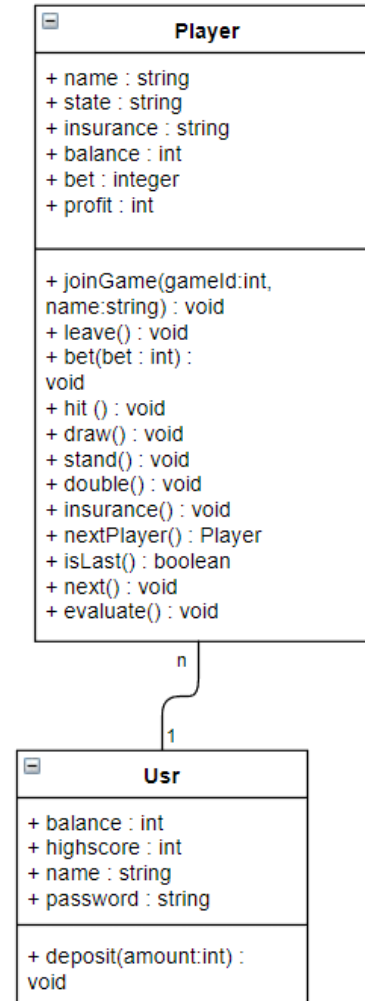
- Für jedes Spiel wird ein Game Objekt erzeugt
- Enthält ID und Gamestate-Informationen
- Umfasst Spielerelemente sowie Dealer
- Funktionalität: neues Spiel erstellen und auswerten

Game
+ id : int + state : string
+ play() : void + evaluate() : void + latestId() : double + reset() : Game



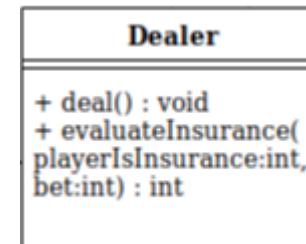
# Usr & Player & Dealer

- Usr für login und globale balance für highscore liste
- Jeder usr erzeugt ein Player Objekt für jedes Spiel dem sie beitreten
- Player speichern Meta Informationen
- Funktionalität: alle Spiel-Aktionen (hit,draw,etc.)



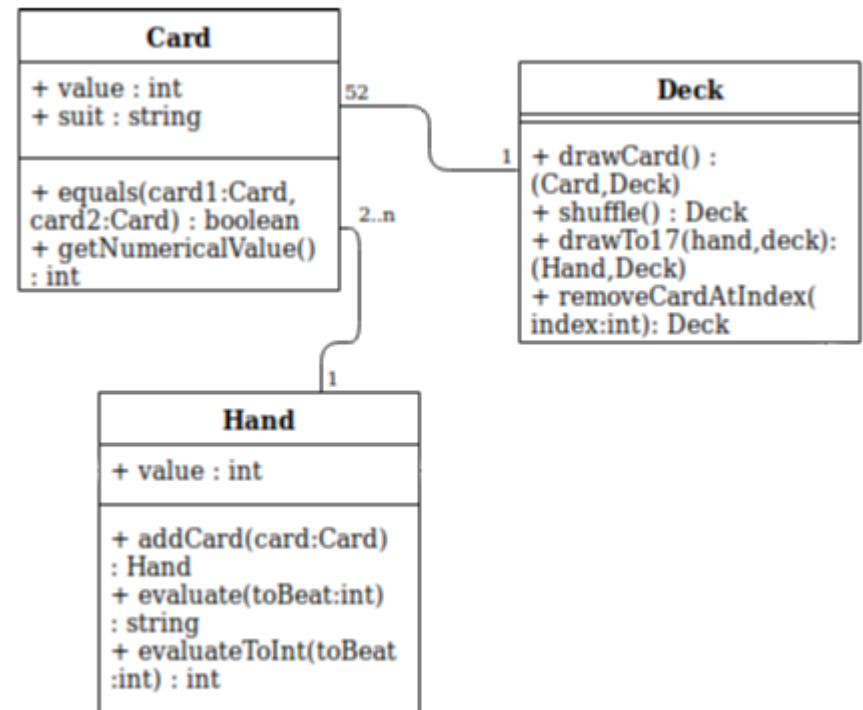
# Usr & Player & Dealer

- Dealer hat das Deck und eigene Hand
- Unterschiede zur Player Klasse
- Funktionalität: Karten austeilen und Insurance evaluieren



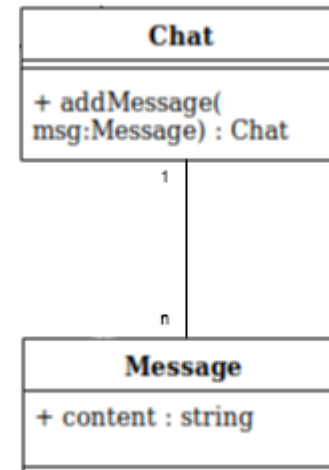
# Deck & Hand & Card

- Deck besteht aus nicht gezogenen Karten („realistisches Deck“)
- Funktionalität: Mischen und Karten ziehen
- Mögliche Erweiterung: mehrere Decks (z.B. Für jeden Spieler ein eigenes)
- Hand hat alle Karten des Spieler/Dealers
- Funktionalität: Auswertung der Karten eines Spielers/Dealers
- Mögliche Erweiterung: Implementierung von Split → 2 Hände
- Card hat Value und Farbe



# Chat & Message

- Extra Feature
- Sinnvoll da Multiplayer über lokales Netzwerk
- Spieler können miteinander kommunizieren
- Information wenn andere Spieler beitreten oder das Spiel verlassen



# XQuery & BaseX

- BaseX als Datenbank
- XQuery als funktionale Sprache die Queries auf XML Datenbanken ausführt
- XQuery selbst kann nur von Datenbank lesen
- XQuery Update Facility zur Kreierung und Änderung von Daten

# XQuery Update Facility

Ziel: angenehmes Spielgefühl mit wenig User input →

Bust und Double beenden automatisch den Spielzug des Spielers

- Keine doppelten replaces in einem return
- Letzter Spieler wählt hit und busted
- Dealer zieht im selben Zug
- Lösung: Dealer zieht vor den Spielern
- Updates von Elementen erst nach dem kompletten ausführen eines return Statement in der Datenbank
- Letzter Spieler doubled oder busted
- Seine Hand noch nicht aktualisiert aber evaluate wird schon aufgerufen
- Lösung: Beim Aufrufen von evaluate mitteilen was die letzte Aktion war und in evaluate beachten

## 4. View

# View

- Anzeigen unterschiedlicher Ansichten
- Unterteilung in Lobby und Game
- Aktuelle Ansicht wird durch die API (Controller) per Pfad bestimmt

# XSL

- Erzeugt abhängig vom aktuellen Pfad und Status eine entsprechende HTML Seite



# HTML






# HTML

- HTML Tags (Table, Button, Form, Input, Label)

GAME ID	STATE	1	2	3	4	5	+
1	DANIELPLAYING	DANIEL(130)	FREE	FREE	FREE	FREE	

NAME	HIGHSCORE	CURRENT BALANCE
 FELIX	160	125
 DANIEL	130	130
 MAX	130	130
4. JANIK	120	120

# HTML

- HTML Tags (Table, Button, Form, Input, Label)
- Verwendung von Submit Buttons für GET/POST Request

Spectating mode

JOIN

◀ LEAVE

Place your bets!

👛 125

BET

Felix

STAND

HIT

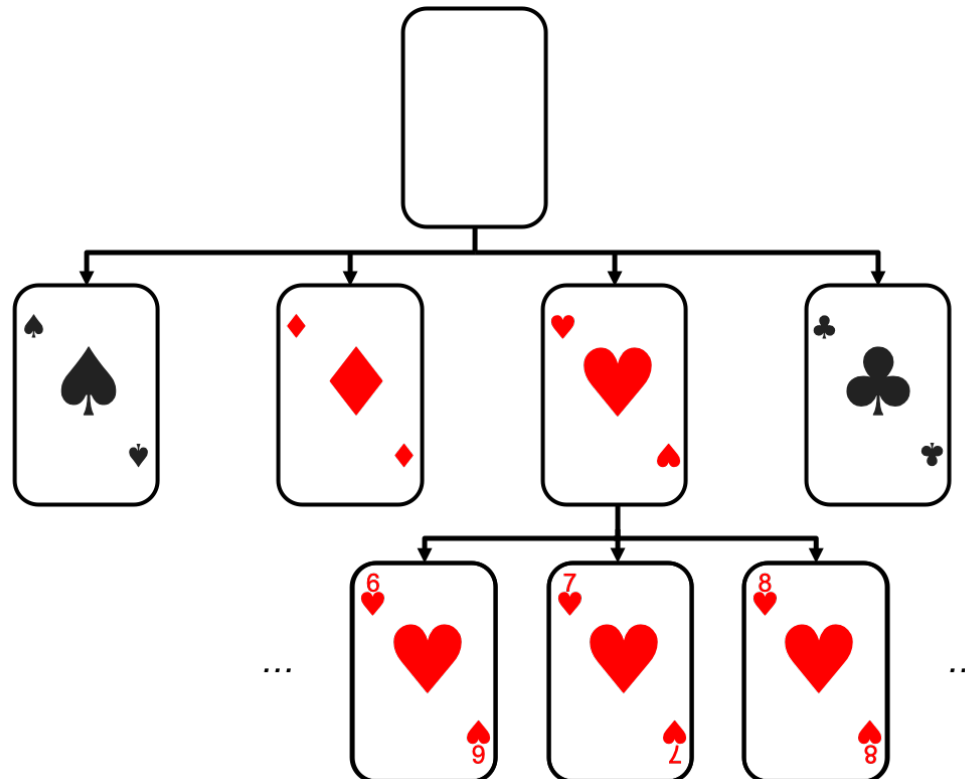
DOUBLE

You won +15!

NEW ROUND



- Bildet den Tisch mit Karten und entsprechenden Spielerlabels ab
- Hierarchische Gliederung der Grafiken für Redundanzfreiheit





- Verwendung von Counter, Advanced Collectors, Transition

```
.chat tr:nth-last-child(-n+8) {  
    display: flex;  
}
```



- Verwendung von Counter, Advanced Collectors, Transition
- BEM Naming Convention

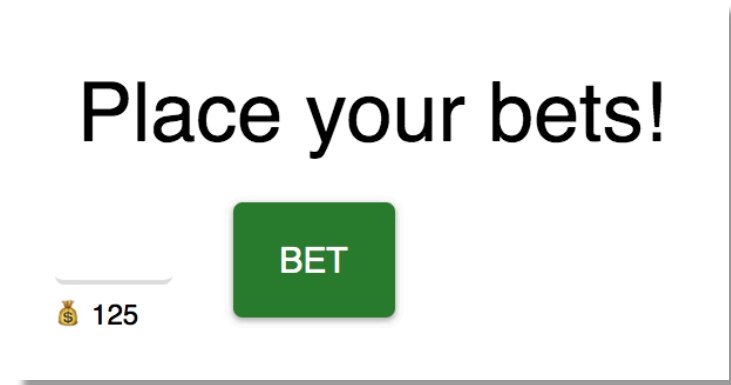
```

/* dialog */
.dialog{
  max-width: 500px;
  background-color: white;
  border-radius: 5px;
  padding: var(--spacing);
  height: auto;
  margin-bottom: 20px;
  box-shadow: var(--shadow);
}

.dialog--header {
  font-size: 35px;
  text-align: left;
  padding: calc((var(--spacing)) - 10px);
}

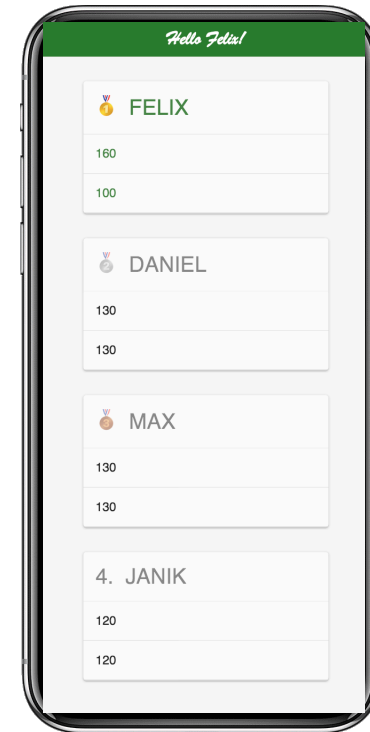
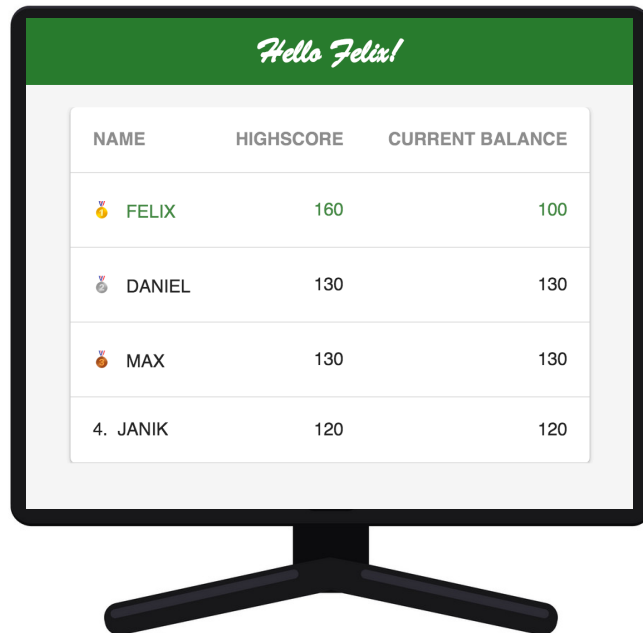
.dialog--content {
  padding-left: calc((var(--spacing)) - 10px);
  font-size: 18px;
}

```





- Verwendung von Counter, Advanced Collectors, Transition
- BEM Naming Convention
- Media Queries für Responsive Design



# 5. Controller



# Controller

- Vermittler zwischen View and Model
- Anfragen von View zu Model (Weiterleitung/Bearbeitung)
- Verbindungsaufbau und Verbindungsabbau



# REST API & RestXQ

GET	/games	List all games
POST	/games	Create new game
GET	/games/{\$gameId}	Open game
POST	/games/{\$gameId}/join	Join game



# 7. Methodik

# Methodik

## Version Control System:

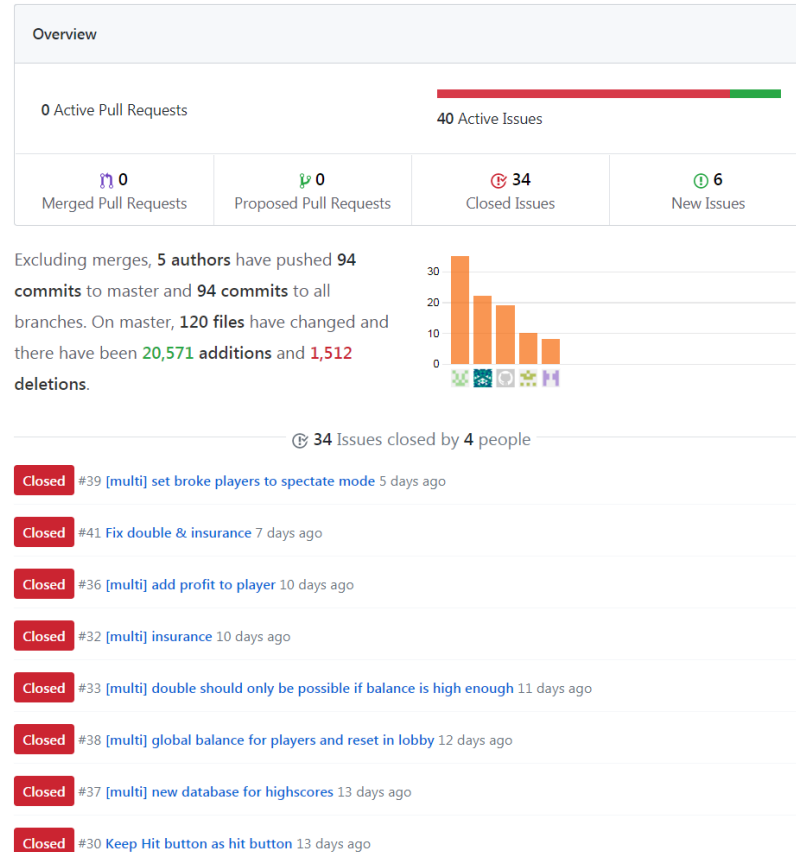
- Git

## Aufgabenmanagement:

- Issues Funktion (Git)

## Fortschritt-Tracking:

- Treffen
- Telefonkonferenzen



## 6. Demo

# Literatur

Vonhoegen, H. (2015) *Einstieg in XML: Grundlagen, Praxis, Referenz*

BaseX Wiki ([http://docs.basex.org/wiki/Main\\_Page](http://docs.basex.org/wiki/Main_Page))

DocBook Wiki (<http://wiki.docbook.org>)

W3Schools SVG Tutorial ([https://www.w3schools.com/graphics/svg\\_intro.asp](https://www.w3schools.com/graphics/svg_intro.asp))

W3Schools XSLT Tutorial ([https://www.w3schools.com/xml/xsl\\_intro.asp](https://www.w3schools.com/xml/xsl_intro.asp))

W3Schools XQuery Tutorial ([https://www.w3schools.com/xml/xquery\\_intro.asp](https://www.w3schools.com/xml/xquery_intro.asp))

RFC 6455 – The WebSocket Protocol (<https://tools.ietf.org/html/rfc6455>)

STOMP Homepage (<https://stomp.github.io>)