

# **XML Technology - SS17 - BlackJack Project - Technische Universität München (TUM)**

**Mr. Robert Urban**

**Mrs. Anne-Catherine Seeser**

**Mr. Lennert Rienau**

**Mr. Michael Mitterer**

**Mr. Manuel Ehler**

---

# **XML Technology - SS17 - BlackJack Project - Technische Universität München (TUM)**

Mr. Robert Urban

Mrs. Anne-Catherine Seeser

Mr. Lennert Rienau

Mr. Michael Mitterer

Mr. Manuel Ehler

---

---

# Dedication

## Important

The whole team appreciates the support and thoughtful guidance given by Prof. Brüggemann-Klein throughout the project and the practical course. Thank you very much for providing some references, help and a very informative and interesting course.

---

# Table of Contents

Development of BlackJack in XML .....	iii
I. BlackJack Project .....	1
1. Introduction .....	3
Project's Context .....	3
2. Technologies .....	4
DocBook .....	4
XML .....	4
XQuery .....	4
SVG .....	4
XSLT .....	5
XHTML .....	5
3. Design and Implementation .....	6
Number of Players .....	6
Card Deck .....	6
Drawing Cards .....	6
Player Actions .....	6
Winning Status and Winnings .....	8
4. Architecture .....	9
MVC Architecture .....	9
Class Diagram .....	10
5. Testing .....	14
II. Organization and Conclusion .....	15
6. Development Environment and Phases .....	17
Development Environment .....	17
Development Phases .....	17
7. Conclusion .....	19
Reflection .....	19
III. Visual Example Workflow .....	20
8. A New Game .....	22
Starting Pages and Initialization .....	22
Bet Round and Initial Cards Dealing .....	23
Players' Parts .....	26
Dealer's Turn and Winnings .....	28

---

# Development of BlackJack in XML

This DocBook contains a short documentation of the design, implementation and testing of the developed BlackJack game. Further, it describes different development phases, the most important requests and responses between the client and a server, the architecture, a class diagram, challenges and solution approaches as well as the organization within the team. In addition, a summary gives a short reflection on the contextual, overall practical course, in which the development was embedded. Last but not least, an example workflow of the application is explained in detail and complemented by images of the GUI, which shows what actions in the game are possible in the different states of the game.

---

# Part I. BlackJack Project

---

---

# Table of Contents

1. Introduction .....	3
Project's Context .....	3
2. Technologies .....	4
DocBook .....	4
XML .....	4
XQuery .....	4
SVG .....	4
XSLT .....	5
XHTML .....	5
3. Design and Implementation .....	6
Number of Players .....	6
Card Deck .....	6
Drawing Cards .....	6
Player Actions .....	6
Winning Status and Winnings .....	8
4. Architecture .....	9
MVC Architecture .....	9
Class Diagram .....	10
5. Testing .....	14

---

# Chapter 1. Introduction

## Project's Context

The BlackJack game documented and presented within this paper was developed by all five team members as mentioned above. The project took place during the summer term in 2017 at the Technische Universität München (TUM) in the context of the practical course "XML Technology" with Prof. Brüggemann-Klein.

One characteristic of this course is, that the challenge is to develop a well-functioning BlackJack browser game, which has to be completely developed by only making use of the wide variety of XML and its related satellite technologies. Hence, the lab was organized that way, that in the first part of the lab, some initial "introductory" lessons were held by Prof. Brüggemann-Klein, in which the students were motivated to get familiar with a wide range of the most important technologies of the XML family.

Afterwards, the second part of the course and its timeline was under control of the individual teams themselves. They were demanded to organize within the group, define rules of BlackJack they want to integrate and finally design, implement, test and document their work. Nevertheless, the teams could orientate themselves on the different exercises and milestones suggested during the first stage of the lab. As it turned out, this was indeed a great orientation for us.

Finally, the third stage of the project was to present the work done and to showcase a demo game of the team's BlackJack version.



---

# Chapter 2. Technologies

## DocBook

As already mentioned above, this project documentation was written in DocBook. The big advantage of DocBook is, that it allows a clear and simple transformation to many different formats - like PDF, HTML and others. Further, it provides an integrated validation mechanism against the related DTD file, which directly leads to better quality, less errors and therefore easier intercorrelation with respect to document exchange.

## XML

As this project is basically all about XML, it is obvious, that this technology also plays a very important role for our project.

XML stands for Extensible Markup Language. It basically is a markup language, which defines a set of encoding rules for digital documents. Having documents encoded into the XML format makes them readable by humans and computers due to its clear and simple structure / format.

Within our project, this encoding format was kind of a starting point. The first step within the development process for example was, to define some states of the later BlackJack game statically within some XML files. Furthermore, this technology is very helpful to understand the needs for a unified exchange format of digital documents and the overall structure, on which the satellite technologies are working on. One example that is the difference between elements and attributes. Last ones should only be used for expressing meta data, which was also clearly under focus for us, in order to build up a straightforward and easy to maintain project.

## XQuery

XQuery is a query language for XML databases, which is specified by the World Wide Web Consortium ((W3C). It is especially appropriate for large databases, as it is very efficient at finding and extracting data from large XML documents.

Regarding our project, Xquery is primarily used within the \*.xqm files. These files are representative for the classes of our overall architecture. Corresponding attributes and functions are mapped accordingly to those files / classes.

The class diagram is added later on in this documentation. Of course, it will also be shortly analyzed, too.

## SVG

SVG stands for Scalable Vector Graphics and enables vector based graphics in XML. Further, SVG is recommended by the W3C. This is one reason, why our team decided to do the visual part of the BlackJack game in SVG.

For the SVG part in general, our team was very thoughtful about to define as many elements within a <defs> element. That way, patterns can be reused multiple times by just referencing them by a unique ID. Those SVG patterns represent an outline or a profile, which can be composed with multiple other SVG objects or images and even be reused in other, composite SVG objects. This is made possible very easily by the tag <use>.

That way, different patterns have been defined for our BlackJack project, like e.g. the players' avatars on the table or the cards of the deck. Especially for the cards, different colors were drawn by combining

multiple geometric forms. Having defined one form once, this form could be reused for printing the color multiple times on the card, or even rotating it on the card.

The form of the BlackJack table was defined by a rectangle with rounded edges in combination with a cut of circle. This model was doubled, whereas the outer implementation was painted black and the smaller one in dark-green with a white margin. For logos, simple pictures can be referenced, for example from the Internet. The rolling form of the insurance lettering and its black edging was implemented with the help of a self-defined path.

Regarding the buttons, it is the case that they can be instantiated in different modes. `<style>` makes it possible to realize different implementations of an object. If the button is active - which basically means that the active player is able to click on it - it is depicted in a certain color. If the button is currently disabled due to the actual state of the game, the button changes its color. Further specifications like reducing the opacity by a simple mouse-over event can be easily integrated as needed.

## XSLT

XSLT stands for Extensible Stylesheet Language Transformation and is a "language for transforming XML documents into other XML documents, or other formats such as HTML for web pages, plain text or XSL Formatting Objects, which may subsequently be converted to other formats, such as PDF, PostScript and PNG. XSLT 1.0 is widely supported in modern web browsers. The original document is not changed; rather, a new document is created based on the content of an existing one. Typically, input documents are XML files, but anything from which the processor can build an XQuery and XPath Data Model can be used, such as relational database tables or geographical information systems. XSLT is a Turing-complete language, meaning it can specify any computation that can be performed by a computer" (Source: <https://en.wikipedia.org/wiki/XSLT>, accessed on 08/18/2017 at around 12:23 AM).

## XHTML

XHTML stands for Extensible Hypertext Markup Language and "is part of the family of XML markup languages. It mirrors or extends versions of the widely used Hypertext Markup Language (HTML), the language in which Web pages are formulated. While HTML, prior to HTML5, was defined as an application of Standard Generalized Markup Language (SGML), a flexible markup language framework, XHTML is an application of XML, a more restrictive subset of SGML. XHTML documents are well-formed and may therefore be parsed using standard XML parsers, unlike HTML, which requires a lenient HTML-specific parser" (Source: <https://en.wikipedia.org/wiki/XHTML>, accessed on 08/18/2017 at around 12:38 AM).

---

# Chapter 3. Design and Implementation

The rules implemented in our project instantiates a more general, less specific version of the game. Nevertheless, all mandatory parts for a general-known BlackJack game have been implemented. Hence, our project's work fulfils all requirements that were prespecified and defined as mandatory in the lab's exercise sheet and goes even beyond that. As an additional note, for all implemented rules the typical and well-known, established versions of official casinos were considered.

## Number of Players

For the amount of players possible to play at one table, we decided to limit the number to five. Of course, it is possible to play a game with less than five players. However, it is required that at least one player is instantiated when starting a new game. Although in most casinos there can play up to seven players at a table, for the scope of this practical course, it absolutely makes no difference if you restrict the amount of players to five or to seven. Hence, due to reasons of better scalability and better overview on a single browser, we decided to go with the five player's version.

## Card Deck

For the overall card deck at a table we decided to go with a package of six decks of 52 French cards each. Also official licensed casinos often work with a package of six decks, in order to harden counting cards. One outstanding characteristic of our cards created with SVG is, that they are mirrored to the middle axis. That way, it is like the real cards and one can easily spin them around without hardening the card identification.

All decks are intercorrelated by a very clear and handy shuffle function. The deck is played until its stack of 312 cards in total (6 X 52 cards) falls beyond a certain threshold. If so, a new, shuffled package of six decks is provided for the table. Further, all cards have bridge-format and depicts extra-large indexes (like Q, J, K, A) for easy identification. Of course, aces can count one or eleven, depending on what is best for the player. This is dynamically evaluated by an intelligent algorithm during run time.

## Drawing Cards

After the initial bet round, each player is dealt out two hidden cards by the dealer. This is done in round-robin method. After the last player (the one to the very right of the dealer) got his first card, the dealer gives himself a first, open card. Following, each player gets his second, open card. Finally, the dealer gets his second card too, but this one stays hidden for now.

During the game, a player can only draw a card when it's his turn. If so, the player can draw as many cards as he likes, as long as he does not exceed the value 21. If that happens, the player loses immediately and is out of game.

At the end of the game, that means all players have been served and are either already out of game or still in the game and wait for the winning status, it's the dealer's turn again. First, the dealer turns up his second, so far hidden card. Afterwards, he must draw cards as long as he is under 17. At 17 or above, he must stand.

## Player Actions

As already implied in the sections above, the players have some basic functionalities when playing a BlackJack game in our project. Due to overview, simplicity and requirements reasons, we kept the

game clear and simple, by just providing the most characteristic functionalities. An overview is given in the following:

- Bet

The bet option is provided to every player before the initial two cards of every player and the dealer are dealt out. Each player can decide, if he wants to bet in this round. Of course, he is free in his decision on how much he wants to bet, as long as he is within the appropriate range. This range is predefined by multiple variables.

Each game, that means each table, has two variables named maxBet and minBet. They defined the range, in which all the bets of the players have to be at this table. An additional variable that constrains the bet values of the players is their individual balance. This balance is generated at the very beginning of the game, where the players can enter their names, too. This balance represents the value of chips, the individual players would have bought at the casino's bank. Another important note in this context is, that the minBet for each table is always at least one. This is an important design decision for our project, as that means that a player just sitting at the table and never betting would be like a player who just blocks a seat at a table. These players are also sometimes taken care of in a similar way in the real casinos, especially when there are other players waiting in a line to play. Thus, this design decision makes our BlackJack game more realistic and also reduces error proneness. For the maxBet we also designed a similar use case, which is that the maxBet can never be higher than one billion (1000000000). This decision is correlated with many other aspects - of both technical and non-technical nature.

For the technical part of this design decision, it is important to consider the ranges of a specific data type. In this case, one has to take care of the integer range. Imagine, only multi-millionaires play on a table like in a James Bond movie. They have almost unlimited money and therefore could easily exceed over the upper integer limit. In order to handle such cases smoothly, a maxBet of one billion is reasonable. For the non-technical part, we made this design decision in order to be as realistic as possible once again. In real casino, there are also some special tables for special guests. That means, there are tables with lower bets and tables with higher bets. Hence, this is exactly the case within our application.

- Insurance

After the initial bet round is finished and all players have received their first two open cards, the dealer got his first open and his second hidden card and everything is set up for the following, actual game, the first crucial player action has to be checked if being available or not.

This action is the insurance action. Whenever the first, open card of the dealer is an ace, the players have the opportunity to take out insurance against a possible BlackJack of the dealer. If the activePlayer decides to do so, he has to pay half his bet for the insurance (in addition to his original bet) against a dealer's BlackJack. If the dealer later on indeed has a BlackJack, he gets back his insurance bet and an insurance winning of the same amount. The original bet is handled separately as described in the following section.

- Hit

The hit action means, that a player wants to draw another card. This action is only possible, if the player is the actual active one, that means, who's turn it is right now. Also, this actions only enabled, if the player has not yet exceeded the value of 21.

- Stand

The stand action means, that the player does not want to draw another card. He can call this action whenever it's his turn and he wants to compete with his current value against the dealer. After a player decided to stand, the next player becomes the active player. An important note for this and the previous actions is, that in BlackJack the players never play against each other. Each single player solely plays against the casino's bank, which is represented by the dealer. Hence, the only one to beat is the dealer.

Concluding this section, all the above explained actions are implemented in the respective functions with the same names. All the functions are handled within the `controller.xqm`. However, the actual logical implementation is within the `player.xqm`. This is in consistency to our overall MCV architecture of the application, as explained in the following chapter.

## Winning Status and Winnings

The winning status of each player is checked at the very end of the game. That means, after each player was the active player and could choose among all the possible actions above. After the last player decided to stand and the dealer draw his cards, all remaining players compare their card values with the dealer's cards value.

Regarding the rules when a player won and lost, we decided to go with the official BlackJack casino rules. The same holds for the winnings. In the following, the different winning situations are explained by generally categorizing into three situations:

- Dealer wins against player
- Player wins against dealer
- Tie between player and dealer

Before going into to much detail, it is important to remember that all players, who played the action hit and exceeded the threshold value of 21 are immediately out of the game. That means, their betting is already paid to the bank and the player loses. Hence, at the status of the game when the winnings are checked, only players are considered, which have a theoretical chance to win against the dealer. This design decision was made due to various reasons, e.g. performance reasons.

As the dealer needs to hit as long as he finally gets a value of 17 or above, the dealer wins whenever his value is higher than the player's one. In this case, the player loses his entire betting. Nevertheless, there can also be some special cases. The first one is, if the dealer is over 21. In this case, the player wins, as long as he is under 21. The second case is, if the dealer's first open card was an ace, the player decided to take on an insurance for half of his initial betting and the dealer indeed got a BlackJack. In that very case, the player loses all his initial betting, but doubles his betting for the insurance. If the dealer got no BlackJack, but still wins, the insurance betting is lost in the same manner.

The player wins against the dealer, whenever his cards value is either higher than the dealer's one, but still in the allowed range (equal or less than 21) or the dealer exceeded 21. In case the player got a BlackJack, his winnings pay off in ratio 3:2 to the initial betting. If the player has no BlackJack, but still wins, his winnings pay off in ratio 1:1 to the initial betting. That means, he gets back his initial betting and winnings of the same amount. These winnings also hold, if the dealer exceeded 21.

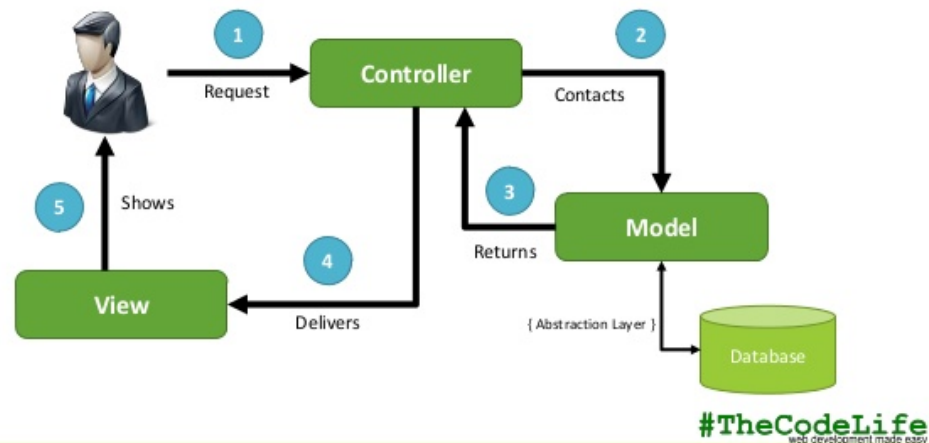
A tie between a player and the dealer always results in the player getting back his initial betting, but without additional winnings. This situation is called a "push" and means, that the dealer's cards value is the same as the player's one (also in case of both having a BlackJack).

---

# Chapter 4. Architecture

## MVC Architecture

### How it works

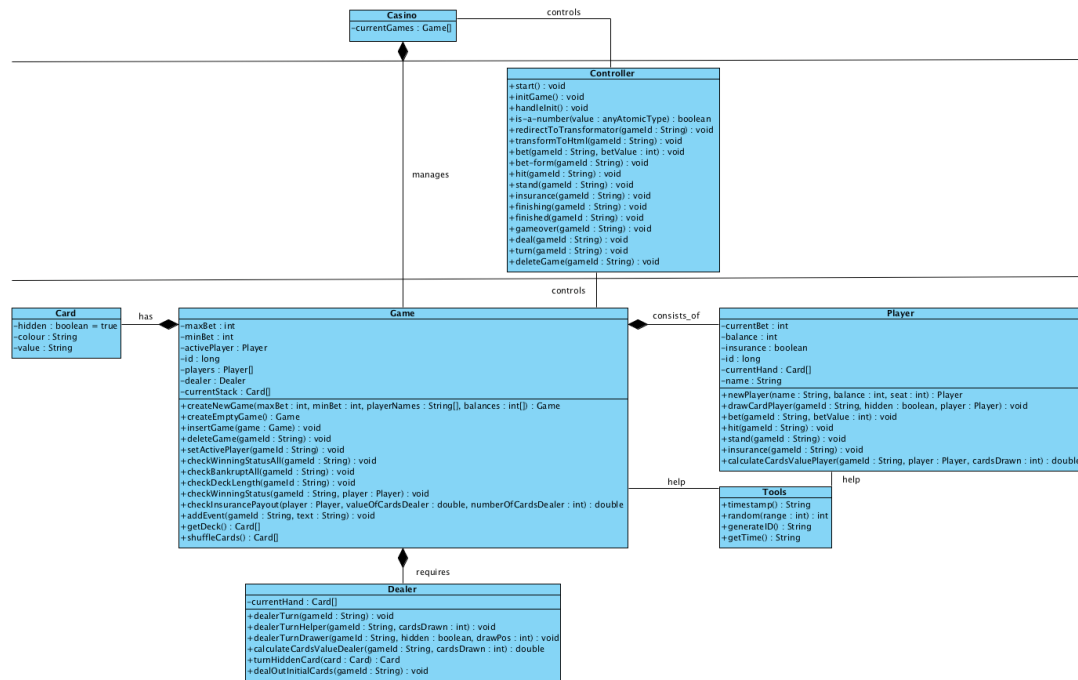


Typical MVC Architecture

The above image is taken from <https://image.slidesharecdn.com/modelviewcontrollermvc-140211001124-phpapp01/95/model-view-controller-mvc-6-638.jpg?cb=1392077579>, accessed on 08/23/17 at around 04:19 PM.

The overall architecture of our application is a MVC architecture as depicted above, which stands for Model-View-Controller. This design decision is very reasonable with respect to the predefined requirements and the fact, that there is database in the background, where all the XML data is stored and a client web browser view, where the client plays and interacts with the web server. Thus, the controller is also mandatory, which is also reflected in the following class diagram.

# Class Diagram



Class Diagram of our Web Application

The class diagram depicted above is a little bit more specific than the general MVC architecture overview. Although it is a more detailed view, one can still clearly identify the three layers of the MVC architecture. The casino server is represented by the casino class. It is the model of the MVC. The controller element, which takes care of handling all requests from the client to the server is modelled and implemented within the controller class. All the other classes are somehow related to the view, as they are generally displayed in the web browser. One example is the class card, which is indeed somehow implemented and shown to the clients in the browser window.

Further, I want to give a short description for the most important methods of our application, which basically map requests to responses between the client and the server and are implemented with XQuery.

- Controller

As already mentioned before, the controller handles most of the client requests by providing REST functions. Furthermore, these functions were mainly very important for testing purposes. As most of the functions' logic is just to call the actual functions, there is not much to explain here.

- Player

- bet

The bet function lets the active player specify his bets before the actual first two cards for the players and the dealer are dealt out. Of course, his betting needs to be within a valid range as described in the sections before.

- hit

Another player action can be to hit. This function takes care of this action, which basically means that the player wants to draw another card. For the adjacent logic workflow it is clear that the sum of all the player's cards has to be counted and compared to the threshold value of 21. Once exceeded, this function is disabled for the active player, as he immediately is out of game.

- stand

The stand option is kind of the opposite of the hit action. A player decides to stand and not to draw another card. Hence, the sum has to be calculated above all cards' values in the following.

- insurance

Last but not least, the fourth and last player action that was implemented of us is to take an insurance, if and only if the first open card of the dealer is an ace. In that case, this action is enabled for the active player at the beginning of his turn.

- calculateCardsValuePlayer

Within this function, the sum of all values of the player's cards is calculated. This is a very important function, as it has to be decided at runtime, whether the value of an ace should count as one or eleven. Many different has to be considered here. One example is, that the game always consists of six card decks at the very beginning. Hence, there could be up to 21 (of 24) aces on a player's hand. Therefore, one can not just generally count an ace as one or eleven. For that reason, the additional parameter cardsDrawn is helpful and the sum without aces is calculated first, before one can consider the gap up to 21 and decide, which aces to count as ones and whether to count one ace as eleven.

- Dealer

- dealerTurn

This function gives action to the dealer. It is very important to understand the construct of it's helping functions dealerTurnHelper and dealerTurnDrawer.

- dealerTurnHelper

As the dealer must draw cards until he is over or equal to 17, the dealer potentially needs to draw more than one additional card. As writing and updating the database needs to be done exclusively, this helper function basically forecasts how many cards the dealer needs to draw from the remaining deck and if he is over 16 together with these forecasted cards. Very elegantly and effectively this tricky process is handled within a recursion, whereas the function calls itself by incrementing the amount of drawn cards, before finally drawing them in the dealerTurnDrawer function.

- dealerTurnDrawer

As described the process in dealerTurnHelper, this function simple draws all the forecasted cards, which were calculated to draw from the game deck. As soon as the dealer is over 16, he must stand. Due to database access limitations, these cards are drawn all together in one function call.

- calculateCardsValueDealer

This function does the exactly same as the calculateCardsValuePlayer, but for the dealer.

- turnHiddenCard

As the dealer's second initial card is always dealt out hidden, this card needs to be turned around after all players were active. This is done within this function.

- dealOutInitialCards

This function deals out two cards to all players and the dealer. Important for this function to represent the actual real casino flow is, that the sequence in which the cards are dealt out is circular from the very left player of the dealer to the very right one, then the first open card for the dealer and finally another round from left to right. However, the second of the dealer is hidden and turned up not yet.



- Game

As depicted in the class diagram, our central class besides the player and dealer class is the game class.

- createNewGame

A new game instance with its own id is created within this function.

- deleteGame

Of course, a game can also be deleted from the database, for example, after it is finished.

- setActivePlayer

This function keeps track of which player is currently the active one. This first active player is always the very left one from the dealer's perspective. After he has finished all his actions, the next one becomes active. After each player was active once, the dealer becomes active and takes out his actions, before the status of winnings are checked.

- checkWinningStatusAll

Checking the winning status for all players is done within this function. That works the way, that all players that are still in game (i.e. the ones, that did not exceed the threshold value of 21) are checked sequentially, after the dealer finished his recommended drawing actions. The sum of each player's cards value is compared to the dealer's one. Depending on the specific situation, the player wins or loses. The concrete regulations on the winning status and the respective winnings is described in the respective section in this documentation.

- checkDeckLength

This function is mandatory, as the decks continuously get smaller while playing. Once a specific threshold is undercut, the six decks get filled up and shuffled again.

- checkWinningStatus

This function is kind of a helper function for the function checkWinningStatusAll. It handles the instance of one particular player. Hence, this function is then called within the checkWinningStatusAll function for each player. Furthermore, the concrete check for how much the player to win or lose is done within here.

- checkInsurancePayout

A very similar function to the one above with respect to its purpose is this one. It checks whether a player took an insurance or not. If so, this value is doubled and returned as winnings in the case the dealer indeed got a BlackJack.

- shuffleCards

A very important function for a casino is to shuffle the cards randomly. Exactly this purpose is fulfilled by this function. Six decks of cards are shuffled all together randomly.

- Tools

The tools class is just kind of a helper class. It provides some general important functions. One such example is the generateID function, which returns a specifically tailored string. This string can for example be used as an id for the games within our casino server.

Of course, there are some other methods, which have not been explained in more detail. However, this is due to the scope of this documentation. Also, only the most important ones are really relevant

to be highlighted here. Thus, focus was put on the important methods, which fulfils all predefined requirements and goes even beyond that.

For additional documentation for all the available functions - not only the above more explained ones - please also consider and take a look at the in-code documentation. Not only the basic purpose for each function is explained, but even the algorithms and some implementation and respective basic design decisions are explained and documented in detail there.

---

# Chapter 5. Testing

Comprehensive testing of our web application was done by simulating requests with specially defined parameters and functions. Many of these functions can be found in the `controller.xqm`. The way they are used is by just sending requests in the URL command line of your browser and see what happens. Effects can then be seen in the baseX database and be compared to the supposed result. Hence, these requests have then be sent to the server and the database, where the respective results could be observed in the XML data. Special focus during testing was put on some borderline cases. This way, for example integer overflows are handled safely. Even further, the game logic sometimes required us to handle special cases safely and realistic at the same time. One example for that were the `maxBet` and `minBet` parameters of the game to play. By repeatedly testing these cases, the likelihood of error proneness could be efficiently reduced to an acceptable minimum.

This procedure was strengthened by differentiating between unit and integration tests, which of course both have been applied in the same way as described above.

As another very important aspect that was considered during the test-driven development cycles, it is important to mention that we as a team always double- and triple-checked the written code of each others by reviews. This was very helpful in multiple aspects. First, this way all team members understood the logic of all other components of the application. Second, we learned a lot from each other by rethinking the implementation style and logic. And third, we could ensure the high quality of our application and its code by eliminating some bugs, which would probably not have been found otherwise.

---

## **Part II. Organization and Conclusion**

---

---

## Table of Contents

6. Development Environment and Phases .....	17
Development Environment .....	17
Development Phases .....	17
7. Conclusion .....	19
Reflection .....	19

---

# Chapter 6. Development Environment and Phases

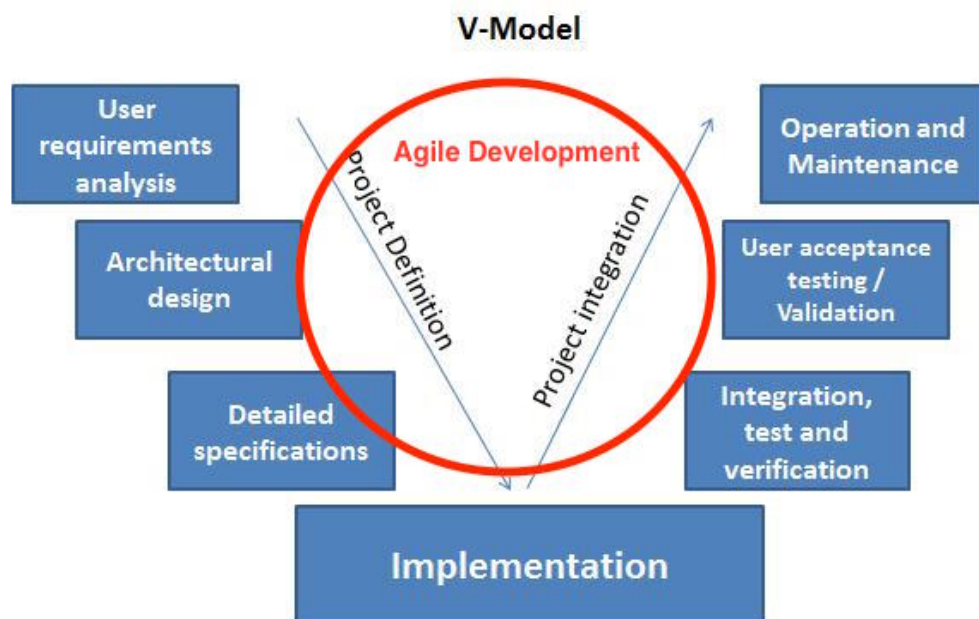
## Development Environment

The common development environment of all team members was the Oxygen XML Editor as proposed by Prof. Brüggemann-Klein. This IDE supports many different formats of files and programming and processing languages like XQuery, XHTML, XSLT and many others. As all team members participated in all stages in the development, it was very helpful, that all team members made use of the same IDE. That way, easy and quick information exchange was possible. Furthermore, we were familiar with the environment, which made implementation and testing even faster.

In order to enable asynchronous development together with synchronized files and code, we decided to set up our own, private team git at GitHub. This worked out very well, as all team members always were on the actual status of the project and we could both easily work on our own part of the code and at the same team commit our achievements in time with all other members. The decision to make the git a private git was due to privacy reasons. This way, we could prevent other teams sniffing at our code and documentation.

## Development Phases

Documenting the different development phases, it has to be differentiated between the overall course structure and the development phases within the team.



V-Modell with Integrated Agile Development

The above diagram is taken from <https://sinergique.files.wordpress.com/2013/10/v-model-illustration.jpg>, accessed on 08/24/17 at around 04:20 PM and represents the overall development phases with the integrated agile development.

For the overall structure of the course, I already described the general workflow at the very beginning of this document. For the first part of this practical course, some introductory lessons were held by Prof. Brüggemann-Klein. These introduced the students into the different technologies and made them

aware of where to go and what the requirements are. The second part of the course was then under control of the individual teams. This was the development lifetime of the application. Last but not least, the third part was to document and present the application.

Throughout the project, Prof. Brüggemann-Klein still took care of all the different teams and coached us in some questions, for example when there were questions regarding the final presentation timeline or similar things. Again, the coaching of Prof. Brüggemann-Klein was very helpful and always in time, which was very appreciated.

As all team members were very interested and enthusiastic about this lab, there were absolutely no problems with the working moral. Hence, this made it very easy for all of us, because no traditionally unnecessary overhead in planning and discussing were needed. Thus, we could focus on the actual development.

The application development was done in a test-driven development style and within cycles. At the very beginning of each cycle, we weekly met up as a group and discussed our status quo, some questions and issues, discussed our reviews done last. Further, next issues, requirements, change request and actual requirements (changes) were discussed in a very professional manner, so that each team member always had to contribute to the discussions. After that, results of our weekly team meetings were published in the git as well as current questions and answers. During the individual or also sometimes synchronized team work during the next week, sometimes single team members met again and worked out on their tasks. Commits to the git were always made shortly before the next team meeting latest, but could also be done anytime in between.

In order to not to go beyond the scope of this document, it can be summarized that our development process can be characterized as a traditional V-model, but with cyclic, test-driven, agile development phases within the embedding V-model. This development model worked out very well for us, as all team members always were on the same page, all were treated equally and all could contribute in a way that was helpful for the team and comfortable for themselves. Furthermore, that way we were able to follow a central theme throughout the project by always having an eye on our requirements, the overall goal, the timeline and the team organization. In addition, due to the inner agile development cycles, we were able to discuss individual ideas and react quickly to any issues that were raised in the team.

Thus, we as a team had a clear and very suitable development lifetime.

---

# Chapter 7. Conclusion

## Reflection

To sum up, the developed BlackJack application of our team fulfils are requirements stated out and predefined. However, our application and the documentation goes even beyond the basic requirements, as we as a team focused on high quality both for the gaming logic and the GUI, to enrich the clients' gaming experience. Of course, enhancements are still possible. As a simple example, one could extend the application to be able to play across mutilple machines. However, this was explicitly not required and wanted, but is just a thought for an additional feature to add, but which goes beyond the scope of this lab.

To conclude, this practical course was an absolutely interesting and informative course. Building a web application solely with XML technology stack is definetly an interesting thing to do. In context with not only developing any application, but a well-known fun game like BlackJack, made the experience of this course even better.

Further, the course was very well structured, organized and coached. Help was given any time when needed. One example for that was, that basically the only major problem for our team was to coordinate across different time zones, as we were distributed some time due to other university liabilities. But with the help of technologies like git and Skype, we could easily overcome this issue and prevent it from becoming a real problem. Once again, the support in that case and with any other minor issues of Pro. Brüggemann-Klein was and still is highly appreciated.

Hence, taking this course is highly recommended for further interested parties.



---

## **Part III. Visual Example Workflow**

---

---

## Table of Contents

8. A New Game .....	22
Starting Pages and Initialization .....	22
Bet Round and Initial Cards Dealing .....	23
Players' Parts .....	26
Dealer's Turn and Winnings .....	28

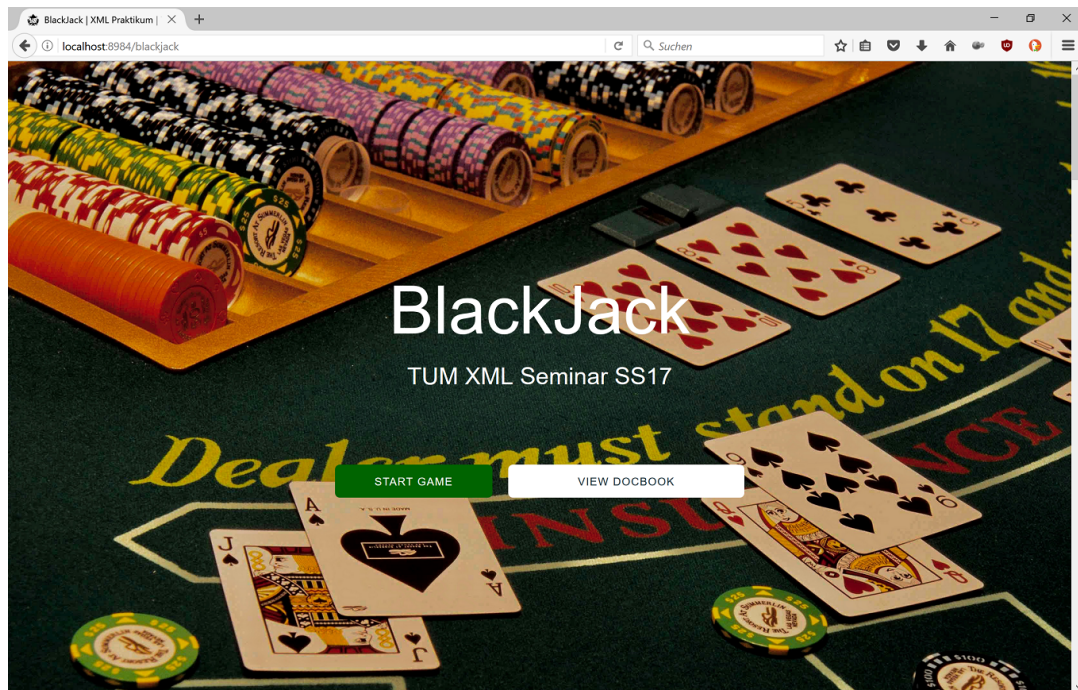
---

# Chapter 8. A New Game

In this part of the documentation, we want to visualize how the GUI of our application works and explain it a little bit more in detail. Hence, we will keep visualizing it with images, which represent the actual application workflow. As you will see, there are always only those actions (requests that are mapped to responses between the client and the server) possible, which actually should be possible in the current state of the game as specified in the requirements.

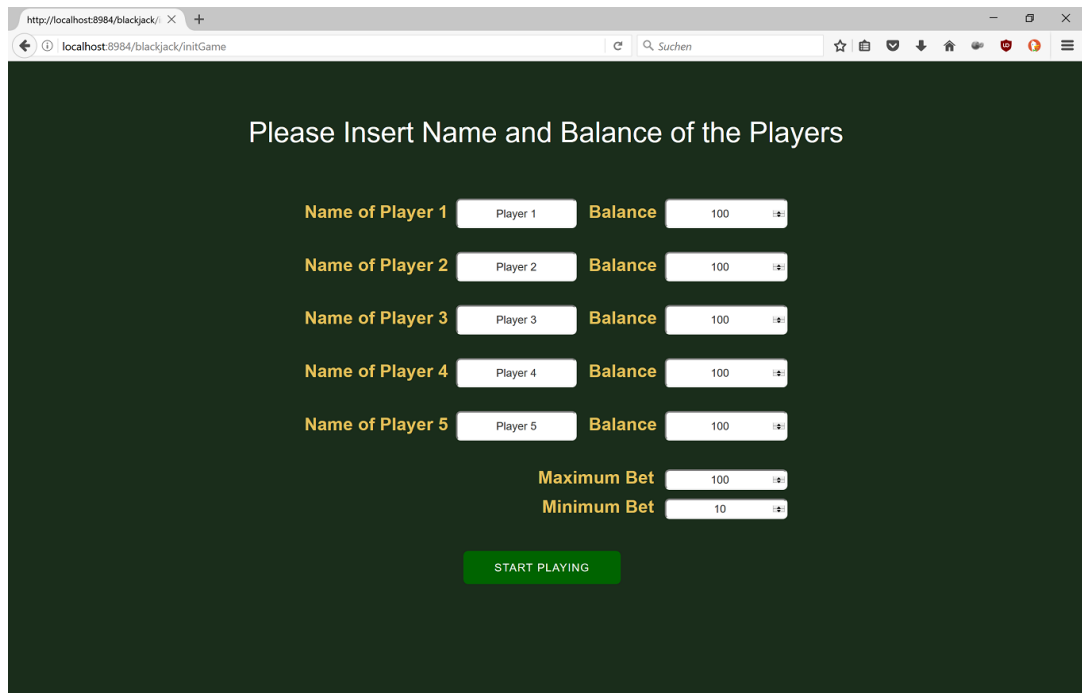
## Starting Pages and Initialization

Calling the game page the guests are welcomed by our landing page. This page introduces into our game, the used technologies and the members of our team.



Startpage of the Black Jack game with our team page

There are links to this documentation and to the GitHub account of each team member. Also, a contact e-mail address is given to get in touch with the developers. To start playing a Black Jack match, the button "Start Game" must be clicked. Now, a form appears where the initial parameters of the game can be entered.

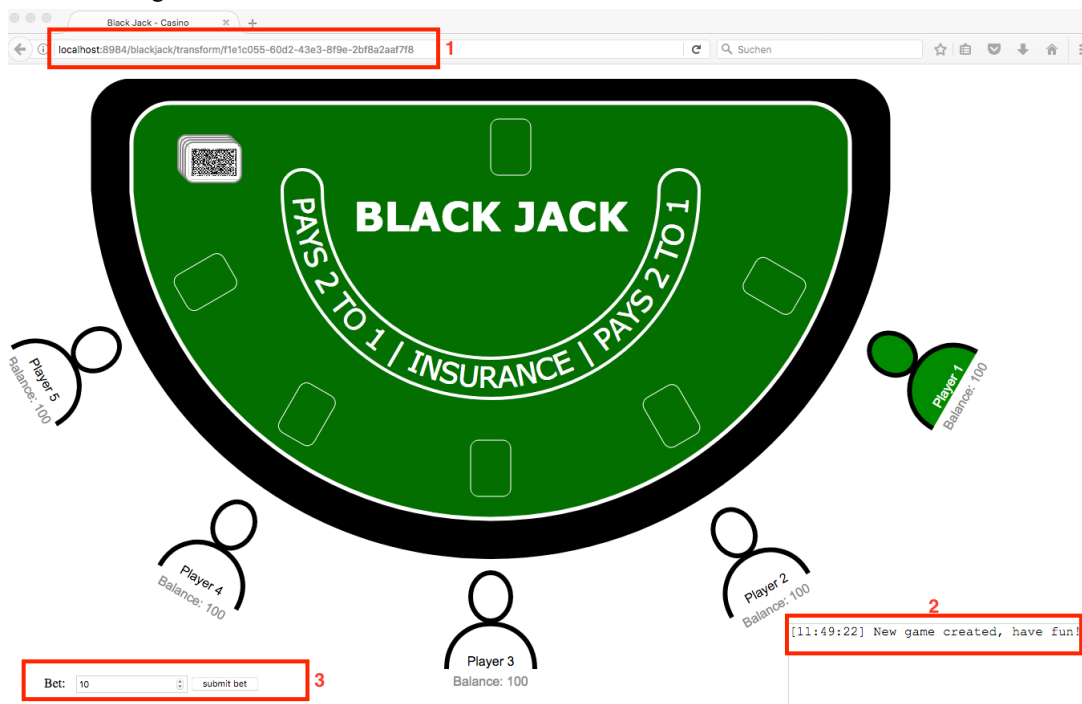


Form for setting the initial game parameters

Here, the game parameter and the players can be specified. There are some example values filled in, but they can be changed free to your wishes. On this page, the minimum and maximum bet can be chosen. Also, the player names can be entered. If an empty player name is chosen or the balance is set to 0, this player will not be in the game. After inserting all data these can be submitted with the button "Start playing".

## Bet Round and Initial Cards Dealing

Once all the parameters of the participating players and the game are submitted, the actual game can be started. At the very beginning of each game, the initial bet round takes place. All players can type in their betting.



## GUI at the Start of the Initial Bet Round

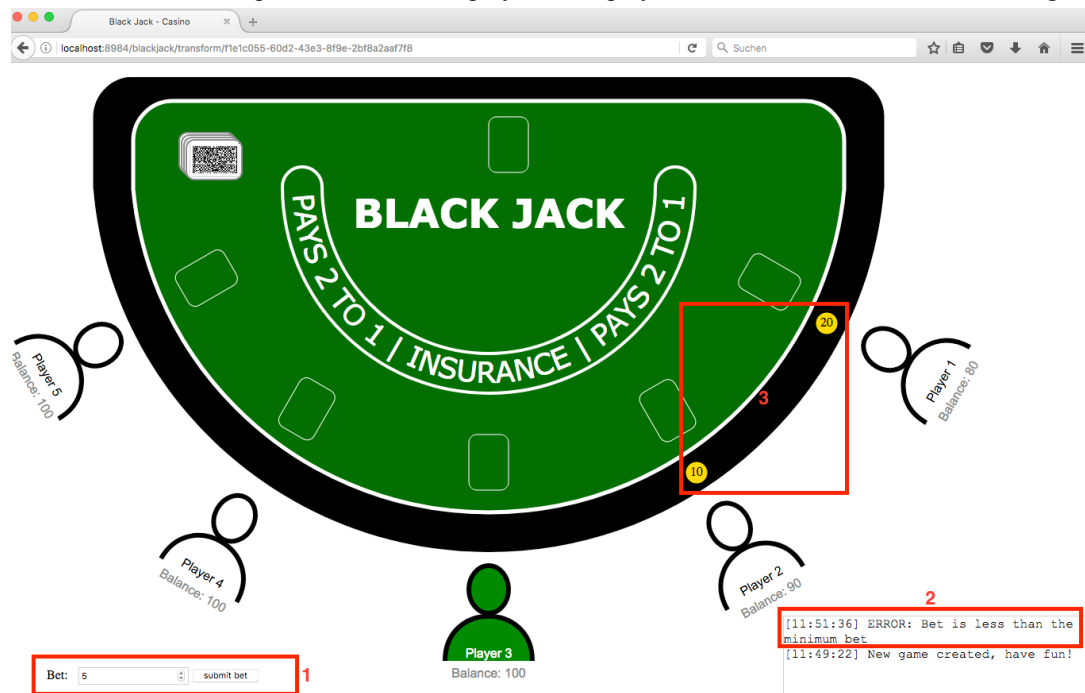
In the screenshot above, one can see the basic GUI of our entire application. The main part of it consists of the game table, where all the chips will be displayed later on. Players are positioned around the table. The first player to always take action first is on the very left to the dealer, so at the very right of the client's GUI web browser. Names and balances of the players are displayed, as well as the hidden card deck next to the dealer's spot.

In (1), we find the URL, that is shown in the browser. As the last part of it, we can identify the current id of the game.

In the bottom right, which is highlighted as (2), we have implemented a short information board, where messages are displayed to the client playing. As we will see in later screenshots, also error messages for invalid inputs are displayed here.

As the current state of the game in our visual example is the beginning of the initial bet round, we can identify a short textbox in the bottom left, where each player can submit his betting. Of course, this input is checked on valid input during runtime after submitting it.

The next screenshot depicts the state, after player 1 and player 2 have submitted their valid betting.

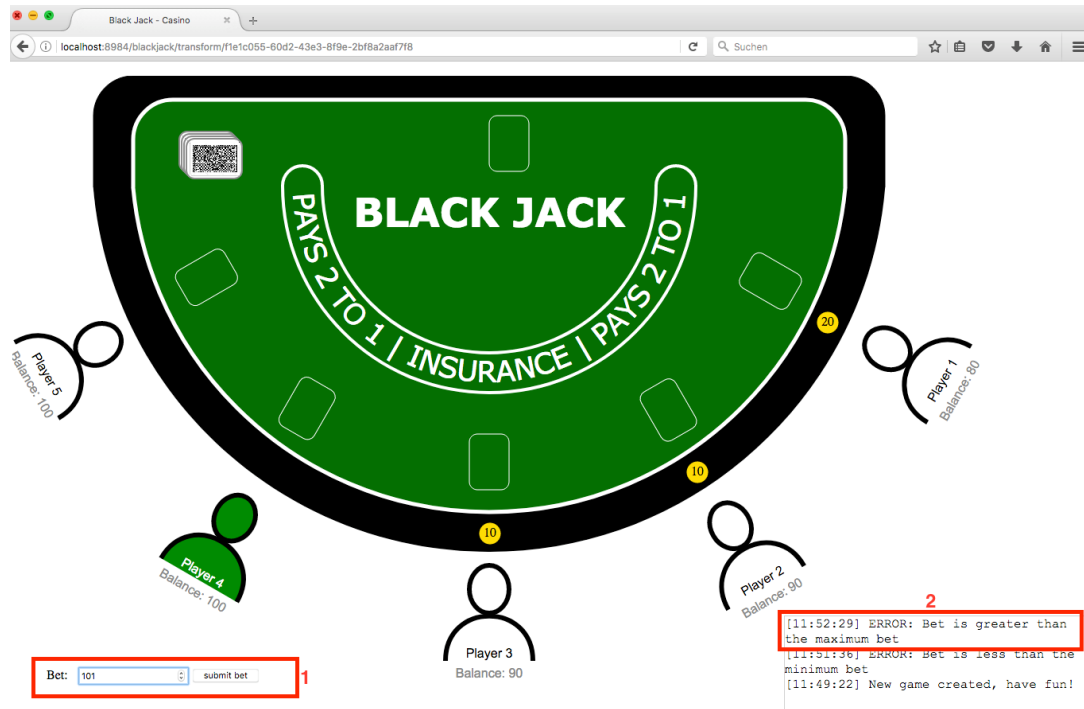


GUI for Invalid Bet by Player 3

These valid betting of player 1 and 2 are highlighted in (3). As shown in (1), player 3 now tried to bet with a value under the minimal threshold value of ten, which was predefined at the starting pages as a parameter for this special game.

Hence, a text message is displayed to the client in (2), which says that the player 3 has to bet at least the value of the minimal bet.

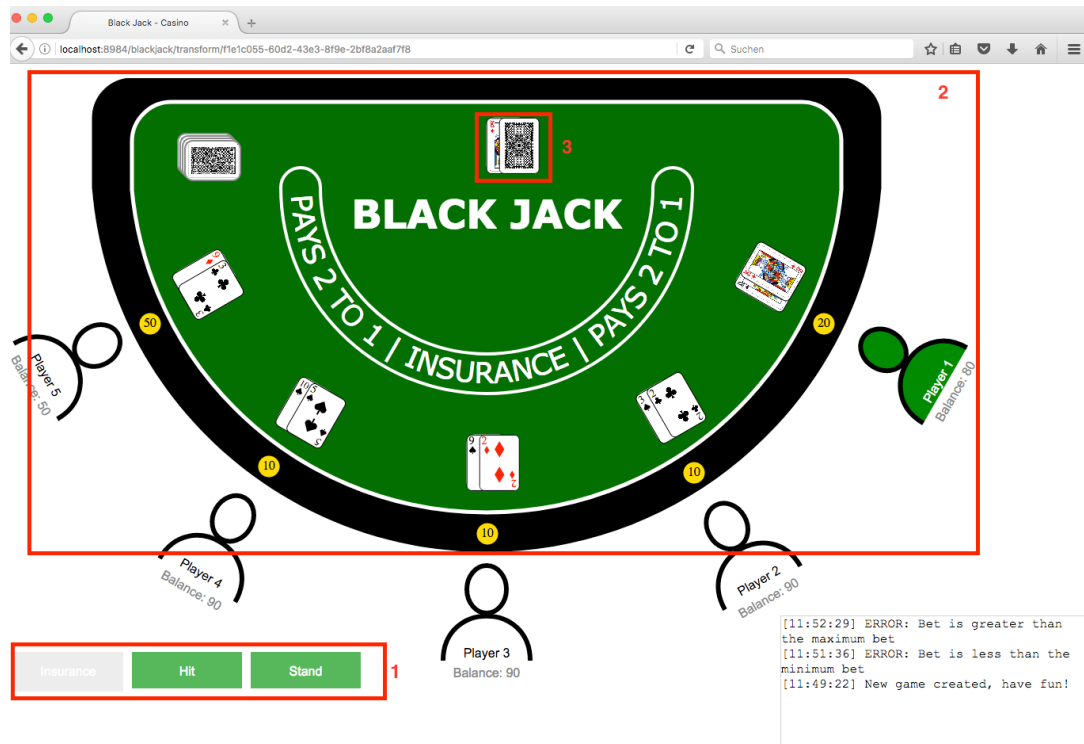
After he did so, the active player indication switches to player 4. So always the green player is the currently active one. In the next screenshot, player 4 tries to submit a bet value higher than the previously defined maximal bet value (which is 100 in this very game).



GUI for Invalid Bet by Player 4

This situation of betting too much is depicted in (1). Again, in (2) a text message is displayed for the client.

After all players have submitted a valid bet value, the initial cards are dealt out and displayed in the GUI. This situation is depicted in the next screenshot.



GUI with Initial Cards Dealt Out

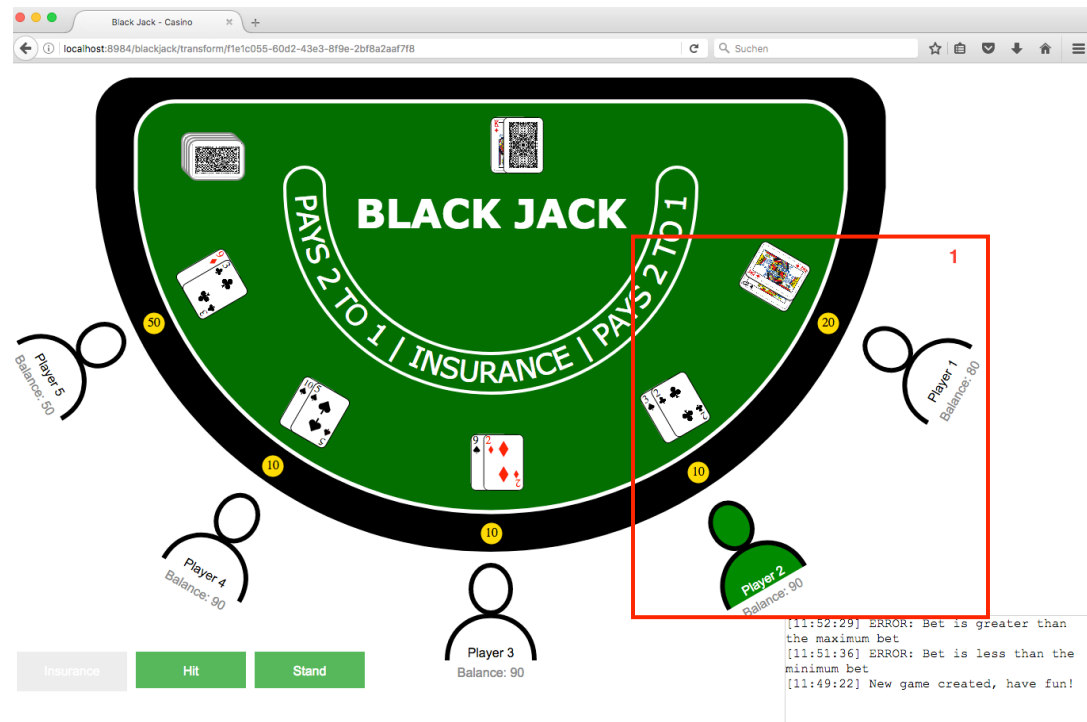
At this state, the bet textbox in the bottom left disappears, as further betting is no longer possible. However, now each player has the option to hit, stand and take an insurance. As the insurance option should only be possible, if the first open card of the dealer is an ace, the insurance button is grey in

the upper screenshot, as it should not be possible as the dealer's first open card is a king. Nevertheless, hit and stand are possible as depicted in (1).

Furthermore, at this state all initial cards are dealt out (2), with the second card of the dealer still being hidden (3). Also, player 1 is the active one again after the initial bet round.

## Players' Parts

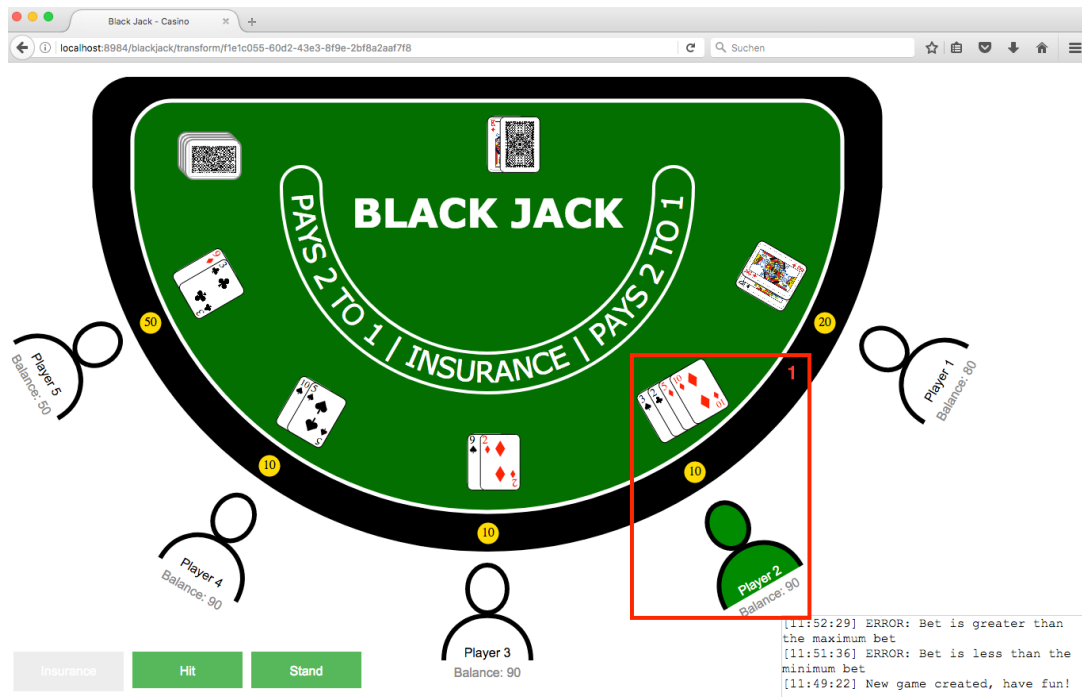
As already mentioned before, player 1 is the first to take action again.



GUI after Player 1 Stands

As player 1 has already a value of 20, he decided to stand on that value. Thus, player 2 became the active player (1).

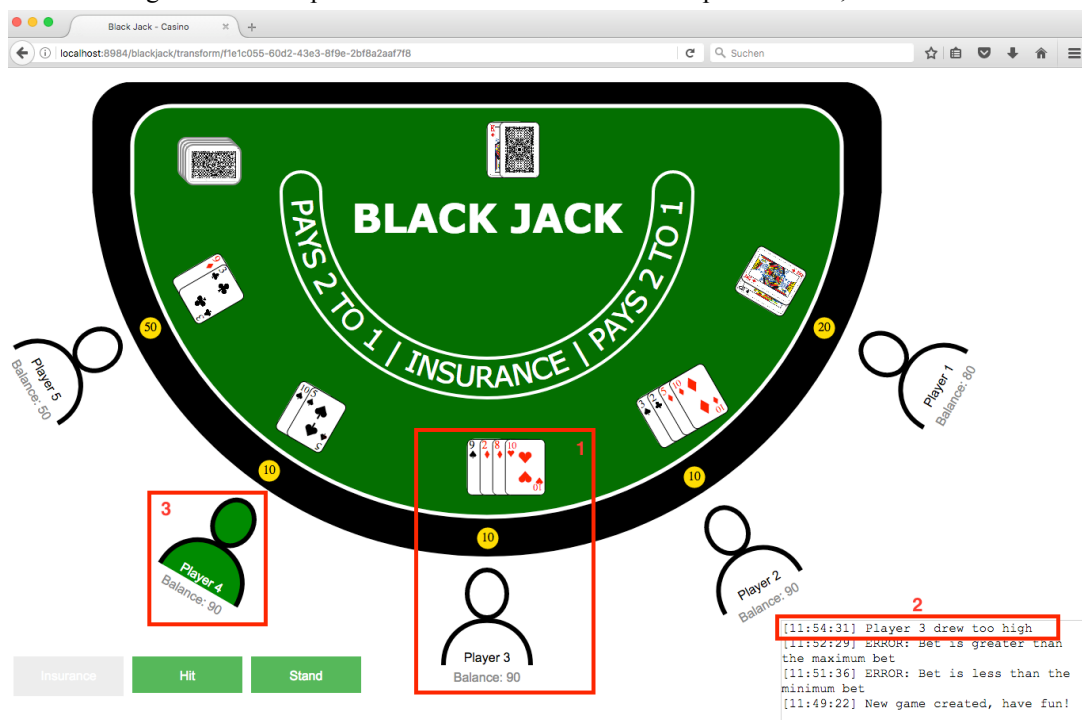
After that, player 2 needs to take action.



GUI after Player 2 Hits Multiple Times

As player 2 only has an initial value of five, he decided to hit three times in a row until he finally received a value of 20 (1). After that, he is able to stand, which makes sense in his case now.

The following screenshot depicts a situation that is similar to the previous one, but different.



GUI Player 3 Over 21

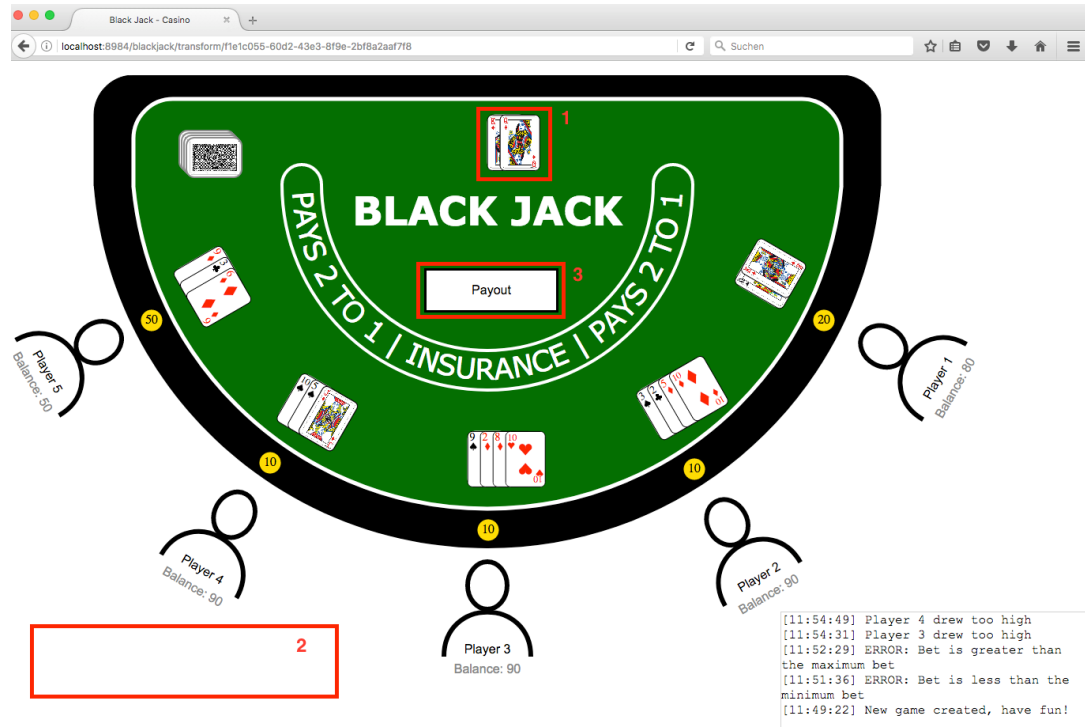
Player 3 got an initial value of eleven, so he decided to also hit multiple times. In the end, player 3 exceeded the threshold value of 21, as he came up with a value of 29 (1).

Hence, he immediately is out of game, a message is sent to the client in (2) and the next player becomes the active one (3).



## Dealer's Turn and Winnings

Finally, after all players took action, the dealer becomes active.



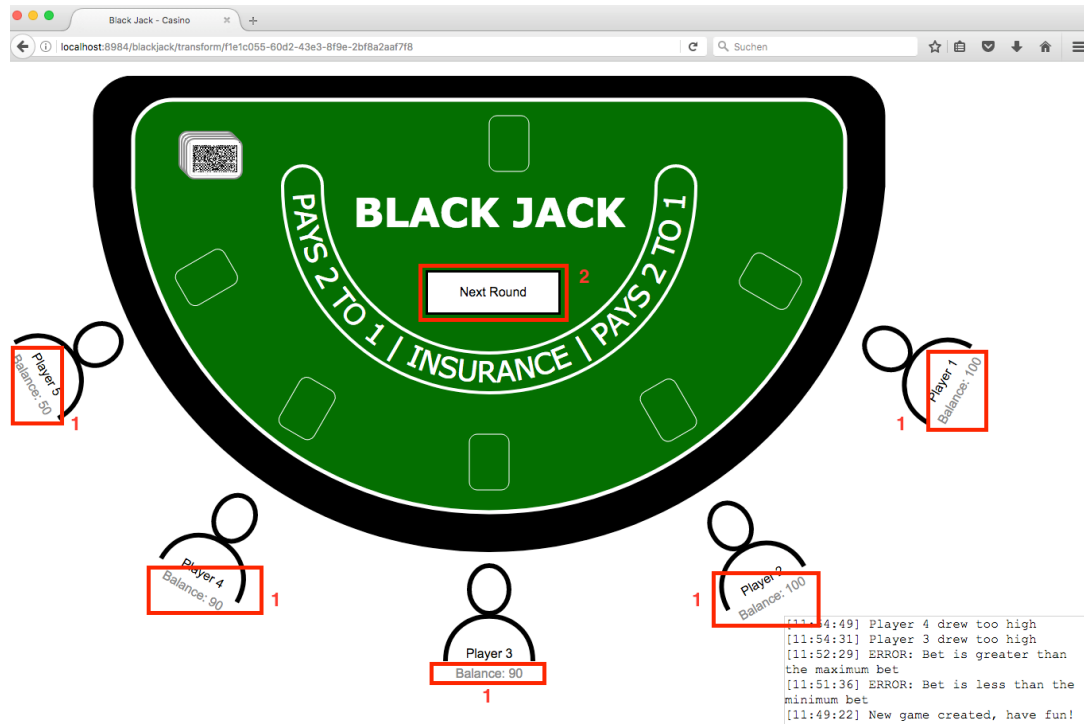
GUI After All Players Took Action

The second card of the dealer is turned up (1), which is a queen in this case. That means, the dealer is over 16 and has to stand, which is why no more cards for the dealer are drawn.

Further, as the dealer does not have to decide, whether to stand or hit, no more buttons for these actions are displayed in the bottom left (2). The rules as described in the previous sections of this documentation describe how the dealer must act. Hence, the dealer is never free in his decision, which means no opportunities have to be provided to him.

However, another button is displayed in the middle of the table (3). This payout button is for finally calculating the winning status of each player, after the dealer had to stand.

What happens after this payout button was clicked is shown in the next screenshot.



GUI After Clicking Payout Button

All cards on the table are eliminated and the players get back their winnings (1). As player 1 and player 2 had the same value as the dealer, they got back their betting without additional winnings. All the other players either exceeded over 21 or lost against the dealer's value. Hence, they lost their betting.

Finally, another button is displayed in the middle of the table (2), which allows to play another round. Hence, this next round would start with another initial betting round and player 1 becoming the active player first.

After having documented one possible game workflow, please keep in mind, that this is just an example and of course there can still be some other options and game workflows. This example was just to get to the point.