BlackJack Documentation

WebSocket Group

Achraf Aroua

Ali Rabeh

Mariia Borysova

Yousri Cherif

Table of Contents

I. Introduction

Project Requirements

Our task during this project was to create an online Blackjack game. The project description contained some functional and non-functional requirements.

Functional requirements:

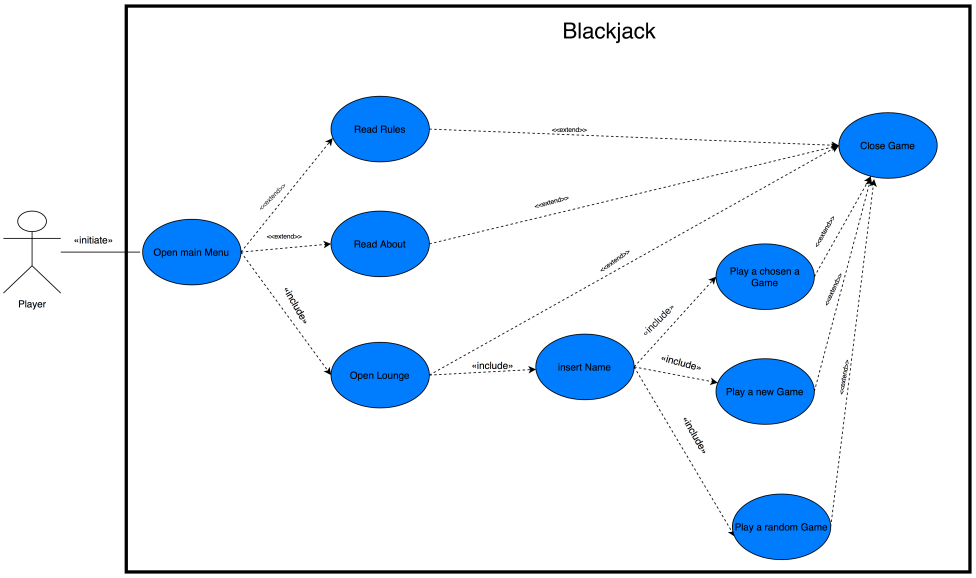
- Multi-client solution (several people should be able to play in their own browser window).
- Lounge where a game can be selected, or a new game can be started.
- Players should have a possibility to register. It is sufficient to choose a name; password is not necessary.
- Lounge has to contain a high score table for the finished games.

Non-functional requirements:

- Blackjack must be designed, implemented and documented as a web application, using only XML technologies (no JavaScript).
- On the server side, BaseX in a WebSocket / STOMP-capable web server (Jetty) must be used to store XML data and to perform XQuery functions.
- To map requests to XQuery functions, annotations must be used in XQuery modules according to RestXQ.
- On the client side, HTML with embedded forms and SVG with the WebSocket element must be used.
- A blackjack server must support parallel games from different clients.
- The documentation should contain installation instructions and, among other things, specify which databases should be created in BaseX and via which URL the game can be started.
- MVC architectural style must be used.
- The documentation should be done with DocBook.

Use Case

Figure 1. Use Case Diagram for the Blackjack Application



Starting the Game

Start screen of the game is called menu and contains 3 buttons. “Rules” button will lead the user to the page where he can find an explanation about the rules in our casino, points system in the leaderboard and will also show the minimum and maximum bets for each round. Minimum and maximum bets are the same for all games and cannot be changed by the user. By clicking the “About” button the player will be directed to a page where the team behind this game will be introduced. By clicking on the “Games” button the user enters the registration screen. There he needs to choose a name which must comply with the format indicated on the screen. Afterwards he enters the lounge and 100$ are assigned to his balance.

The lounge

There are 2 tables in the lounge – “Leaderboard” and “Games”. In the leaderboard the names and the points (score) of the TOP-8 players of the casino are listed by descending order. When a player first

enters the game, he has 0 points. The second table contains a list of all of the ongoing games. In a casino there can be from 0 to 8 simultaneously running games. If there are less than 8 games, the player can create a new one by clicking the “New Game” button. The player can join any ongoing game. If all the places in the game are occupied, he will watch and wait until a seat becomes free. There is also a “Random Game” button in the lounge that will let the player join a random game. Once a game is created, it cannot be deleted.

The Game

There is no strictly defined set of rules for blackjack. In fact, they differ from one casino to another. Below are the rules that our team set for our variant.

Up to 5 players can take part in one game. Blackjack is played against the dealer. Even if there are several players at the table, they all play against the dealer, and not against each other. Therefore, the main goal is to collect a combination of cards that is superior to the dealer’s hand, but without exceeding 21, also called busting. The value of a hand in blackjack is determined by adding up the values of the player’s cards. Any combination of the cards that sums up to 21 regardless of the number of cards is called “blackjack” in our casino. The best hand has a sum of 21. Any combination with a total score of more than 21 is automatically eliminated from the game.

The player does not always need to get 21 points in order to win against the dealer. To do this, it is enough to have a hand, the sum of which exceeds the total amount of the sum of the dealer. At the beginning of the game, players receive two cards face down, and the dealer puts out his cards as follows: one card face up, and the second down. Dealer must continue drawing cards until he reaches a total of 17. During each round one full deck of cards is used (52 cards).

The Bets

Before the start of the round, all players make bets in turn from right to left. The minimum bid is $10; the maximum is $100. The active player is identified by a pointer above his card zone. A player can leave the game without losing funds only before making a bet. The game uses chips in denominations of 1, 5, 10, 25, 50. After placing the bets all the players and the dealer get the cards.

Card Values

All cards with faces (king, queen and jack) bring 10 points. An ace can cost either 1 or 11 points, depending on which value is more profitable at the moment. The cost of the remaining cards is equal to their numerical value. Card suits have nothing to do with their value.

Player’s actions

- Hit - add another card to the player's hand. The player can ask for as many cards as he wants, as long as their amount is greater than or equal to 21. If a player received blackjack immediately after the bet, he can still click “Hit” and continue playing.

- Stand - this tells the dealer that the player is refraining from further distribution.

- Double Down - player doubles his initial bet if the value of his cards is less than 11, but after that player will receive exactly one card.

- Surrender - after the initial distribution, the player can choose to continue the game or to give up and pick up half of his original bet. Only available before any other action is performed.

Possible outcomes

- If the amount of player’s cards exceeds 21 points - the bet is withdrawn in favor of the casino.

- If the amount of player’s cards is exactly 21 points (“blackjack”) – the player automatically wins, no matter what cards the dealer has. Blackjack pays 3:2 to the initial bet.

- If the sum for player’s cards is less than 21 points, but more than the sum for the dealer’s cards; or if the dealer busts, the player wins. Win is paid 1:1 to the initial bet.

- If the amount of player’s cards is less than that of the dealer, the bet is withdrawn in favor of the casino.

- There are no draws in our casino. Thus, if the sum of the player's cards matches the sum of the dealer’s cards, the player wins and is paid 1:1.

End of the Round

After the end of the round, any player can start a new round by clicking on the corresponding button. If a player does not have enough money for a minimum bet, he will be automatically excluded from the game. After that, if another player is on the waiting list, he will be able to take a free seat. A quitting player can enter the lounge by creating a new name, or by using the old one. In the second case, he will receive a new 100 $, but his points in the leaderboard will remain the same as before.

Leaderboard points

The number of points in the table corresponds to the number of money won/lost for all rounds played by the player in all games. Points can be both positive and negative.

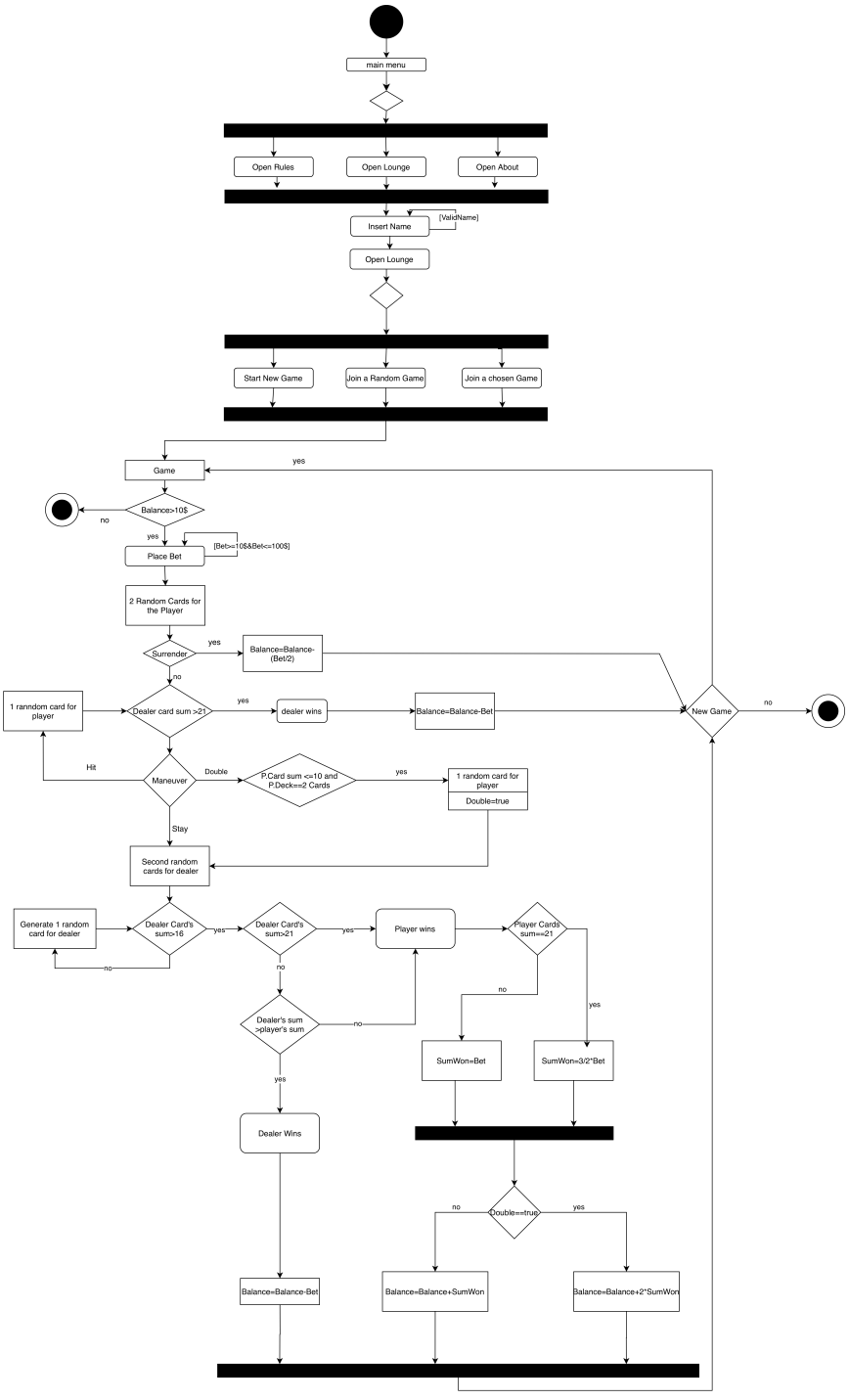
Other actions:

- Insurance. By opening an ace at the dealer during the distribution, the player can prevent probable blackjack. The level of insurance is equal to 50% of the rate made initially. If the dealer has blackjack, the player’s bet will be returned, and the game continues as usual. This function will not be implemented in our casino.

- Split. A hand is split into two if both cards have the same value. If both hands win, the player gets a double win. If both lose, the player loses twice. Split is done by pulling the cards to the sides and repeating the original bet. This function will not be implemented in our casino.

Following activity diagram represents the complete flow of the game. Following Use Case diagram shows interactions between the user and the application.

Figure 2. Activity Diagram for the Blackjack Application



WebSocket Team

Our team is called WebSocket (ws) and consists of 4 students:

- Achraf Aroua (Informatics, Bachelor 6th semester)
- Ali Rabeh (Mechanical engineering, Bachelor, 8th semester)
- Mariia Borysova (Informatics, Bachelor 6th semester)
- Yousri Cherif (Informatics, Bachelor 6th semester)

We decided to use WhatsApp for communication between the team members. We had regular meetings approximately every 2 weeks at the university before Christmas to divide tasks from the exercise sheets and discuss problems. Our team presented the solution for the second exercise sheet “SVG” on the 14th of November. After the winter examination phase we started to hold weekly online meetings via Zoom due to the Coronavirus outbreak. To share our implementation between the team members a Github directory was used. To share the presentation slides, notes and other written materials we created a Google Drive directory. For the implementation all of the four team members decided to use an IntelliJ IDEA IDE.

Every team member wrote a part of the code and a part of the documentation, but main responsibilities were divided as follows:

- Achraf was responsible for the functionality of the game.
- Yousri was responsible for the design.
- Mariia and Ali were responsible for the preparation of the documentation, the presentation and the testing.

II. Modeling and Implementing

Client-Server Architecture

To model and implement our web application we used a client-server architecture. This means that the client runs in the web browser, the server runs in the web server (Jetty) and these two components communicate with each other using the HTTP protocol. The server contains the application code that manipulates the data and the database in which the data is stored. Only application code can access the database and the database is architecturally one level below the server. That’s why we get a typical three-tier web application architecture. To the Tier 1 belong any number of browsers through which users interact with the game. Web server is situated on the Tier 2 and a database system is Tier 3. Tier 1 and 2 communicate via HTTP requests and responses, Tier 2 and 3 communicate via XQuery methods. There is no direct connection between Tier 1 and 3.

MVC for a Single-Client Web-Application

One of the project’s requirements is to implement the application using the Model-View-Controller architectural style. We used an MVC with Passive View variant to implement our application. This means that Model and View can communicate only via Controller and Model will not directly notify View about the updates.

Task of the View component: displaying the information to the user (state of the application), tracking the user interactions, sending the user’s responses to the Controller, waiting for the response of the controller and displaying the changes to the user. The data exchanged between the View and the Controller uses XML for the representation.

Task of the Model component: storing, providing access and manipulating the data. Methods in Model are called by the Controller and it’s a single opportunity for the Model to change its state.

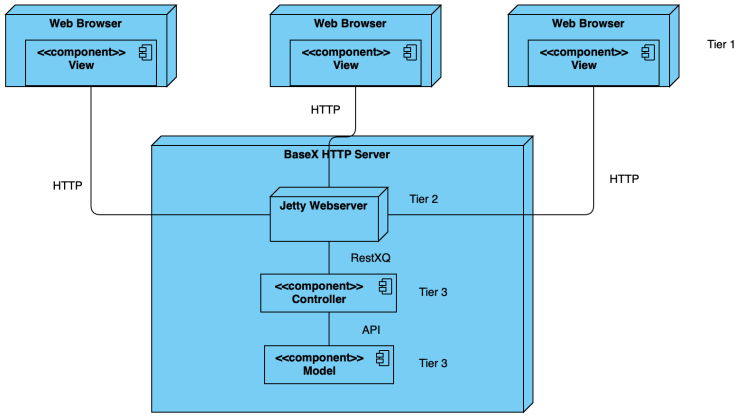
Task of the Controller: Controller is a bridge between the View and the Model. It receives the information about all of the user’s interactions, calls the methods in Model (if needed), gets the information from Model, converts it to the information for View and sends it back to View, so the View can display a new state of the application to the user.

Mapping MCV to the three-tier architecture

Since our application is an XML-based application, both Controller and Model are XQuery-Modules and are run in Tier 3 by a single backend XQuery processor (in our case BaseX Database system). Thus, Controller and Model are situated together on Tier 3 and communicate using the XQuery methods.

Web server on the Tier 2 is a generic component and is responsible for HTTP requests from users and HTTP responses to users. Controller has to translate the HTTP requests to the XQuery functions. Rules for this translation are defined by RestXQ annotations within Controller. Controller also needs to convert XQuery back to HTTP to create a response (the features of this process will be described later in the chapter “Implementing the Model component”). There is no application-specific software on the Tier 2. Tier 2 and 3 are both run on the BaseX HTTP Server. MVC architecture for single-client version with distribution at various Tiers is shown at the deployment diagram below.

Figure 3. MVC Deployment Diagram for a Single-Client Application

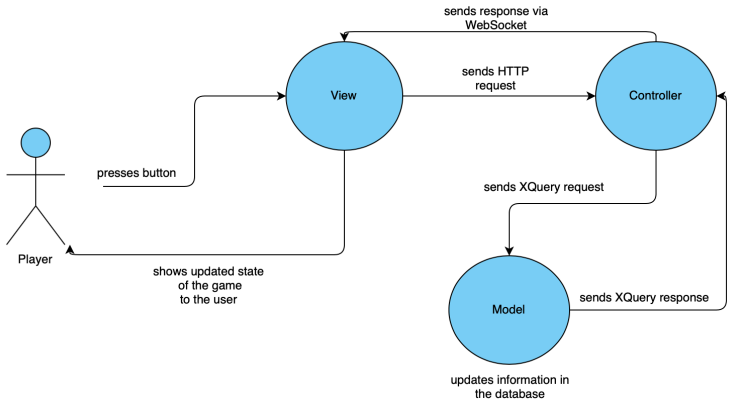


WebSocket

One of the main requirements for the game was to make it multi-client. By using the regular HTTP, the client will get the response from the server only as an answer to its prior request. That is why, for example, the screen of the Player1 will not be updated after Player2 finishes betting. Websocket is an HTTP extension protocol, which allows us to create a bidirectional communication between the client and the server. Now the server has a possibility to send a response to the client without the prior request (do a server push). The server must use a STOMP (Simple Text Oriented Messaging Protocol) on top of the Websocket.

The Use Case diagram below demonstrates how the player now interacts with the application and how information is transmitted within the system.

Figure 4. Use Case Diagram for the MVC with WebSocket

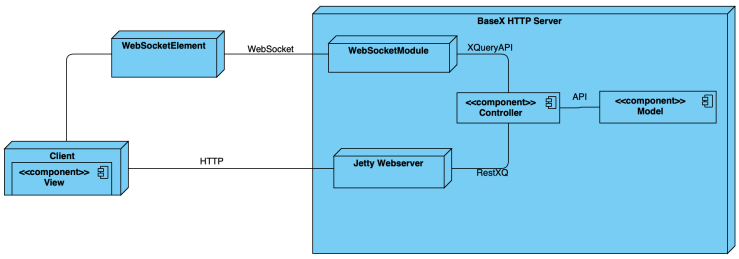


Changes in architecture compared to the single-client version

WebSocket protocol is used for the responses from server to client instead of HTTP.

MVC architecture for multi-client version is shown at the deployment diagram below.

Figure 5. MVC Deployment Diagram for a Multi-Client Application



Modeling and Implementing MVC

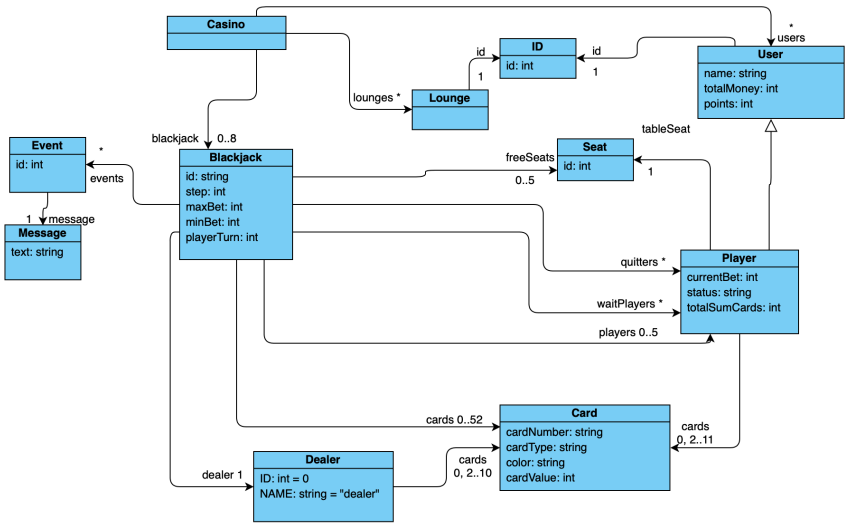
SHORT INTRODUCTION:

The word **LOUNGE** is used to describe the screen that contains games and leaderboard. The word **LOBBY** is used to describe the screen containig the three buttons “Games”, “Rules” and “About”. Word **LOBBY** is used everywhere throughout the implementation except for the game interface, where we decided to use the word **MENU** as it seems more user-friendly to us. Thus, words **LOBBY** and **MENU** are used synonymously throughout the implementation and documentation.

Modeling the Component Model

UML Class diagram below represents the structure and the relationships between the objects as they are stored in the database. To make the diagram more understandable, we divided it into two parts. The first contains only attributes, and the second contains only functions.

Figure 6. Class Diagram for the Model Component (attributes only)



The main element of the application is Casino class. Casino can simultaneously contain from 0 to 8 `games` and the list of all of the registered `users`. The casino also contains the list of `lounges`. Each lounge displays the information to one specific user; therefore, lounge and user are connected and share the same `id` attribute. Whenever the user enters the lounge, he is stored there. Whenever the user leaves the lounge, he is removed. It was modeled so that we could know who is in the lounge at

the moment and could send them an update via WebSocket whenever an update occurs. For example, when one user creates a new game, all users in the lounge are notified.

The next important part of the application is the game itself. Class for the game is called Blackjack. It contains the following attributes: `id, step, maxBet, minBet` and `playerTurn`. The `step` attribute controls the current state of the game and can have three values: “bet” when the player is betting, “play” when players are playing or “roundOver” when the round is over. Besides, Blackjack has one `dealer`, one deck of `cards`, which is full at the beginning of each round and three lists of players. First one – `players` are the players currently participating in the game. Second – `waitPlayer` are the players currently watching the game and the last one – `quitters` are players who are leaving the game. If a player pressed the exit button or lost all of his money, he will be added to the `quitters` list. This is necessary for sending the player to the endGame screen via Websocket instead of the game table (Blackjack screen).

Blackjack and Player are also connected via the Seat class. The game keeps track of the `freeSeats` and the player has a `seat` attribute that assigns a particular place on the table to the player.

Dealer class has attributes `id` and `name`, which are predefined and will not be changed during the game. Dealer and Player can either not have `cards` at all (during betting) or have from 2 to 10 (for Dealer) or from 2 to 11 (for Player). Class Card have attributes `cardNumber, cardType, color` and `cardValue`.

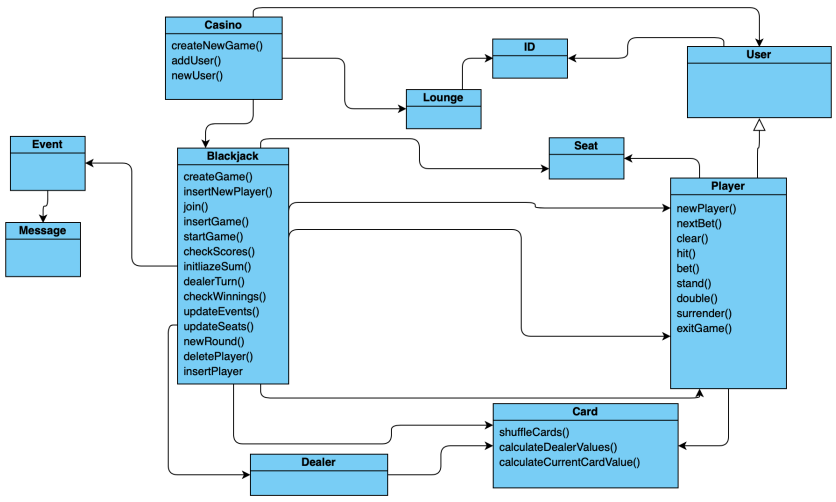
User class is a parent of the Player class. They have the same `id, name, totalMoney` and `points` attribute. Actually, the user is a player in the lounge. After the user enters the game, he gets additional attributes `currentBet, status, totalSumCards` and turns into a player. `status` identifies whether the player lost, won, surrendered or got a blackjack.

Last two classes in the diagram are Event and Message. `events` are an attribute of Blackjack and this attribute is necessary to show the messages to the player. Event has an `id` which defines the player to be shown the message and if the `id` of the message is 0 then all players will get the message. Message class contains `text` of the message itself.

To make the game dynamic, we added numerous XQuery functions in different **.xqm** files.

We tried to assign each function to different classes based on the UML-Class-diagram to make the game more consistent and clear. For example, the function `shuffleCards()`, which is a function that returns a shuffled deck, is part of the Card class. However, not all functions are mapped into classes, the helper function can be directly called from each other function and just useful to compute an output (e,g a random number). Getters are omitted too. Next class diagram represents our mapping of the functions to classes.

Figure 7. Class Diagram for the Model Component (methods only)



Full list with all of these functions can be found in **Appendix A**.

Basically these functions are reacting to some sort of event. There are two types of functions. Functions of the first type react by making an update to the model (like for example `insertNewPlayer()`) and functions of the second type return a static result like XML-Node or an HTML-page (for example, `createGame()`). Functions of the second type do not operate on the existing object and act like constructors in other programming languages.

Implementing the Component Model

Classes of the UML Class diagram are stored as XML objects in a BaseX Database.

Functions are divided into separate files for easier and more clear structuring of code. The file **action.xqm** represents the actions that players can perform during the game (e.g. bet, stand etc.). The **cards.xqm** collects the functions that are related to the cards and to the calculation of their sums. The file **player.xqm** is needed, for example, to create a new user and player. **Game.xml** maintains operations to create, launch and conduct a game. **Helper.xqm** contains secondary functions.

All of the methods from the table above are implemented as an XQuery method and some of them use the XQuery Update Facility extension. Methods are executed by BaseX. Usual Xquery method can either return a value or make an update in the database, but not both. This is known as XQuery Update Constraint. However, most of our methods require a return value and an update within a single query. That is why we use the `update:output` method in our functions. This method gets the return value of the XQuery function and uses it to update the state of the database.

We also used web-redirect to resolve the constraint. Using it we can go from one RESTXQ function to another. We used it after `update:output` to make one function call another function after the update.Thus, to redirect the user after updating we used `update:out(web:redirect(URL_of_another_function))`.

After updating the Model we want to send a response back to the user. We need to go with the Xquery Update Facility, thus, we call a method that generates a response to users using XQuery. In our implementation the `draw` and `showGames` methods are responsible for it. First one generates a response when the player is in the game and second one generates a response when the player is in the lounge.

Modeling the Component View

View component contains a number of different screens. In the following table we modelled all of the important information about each of them. Table describes all of the specific attributes that are needed to show information to the particular user and the interaction that are offered to the user. Last column contains the information which View sends to the Controller using the HTTP Request. The lobby can be accessed through the web browser via **localhost:8984/webSbj/lobby**.

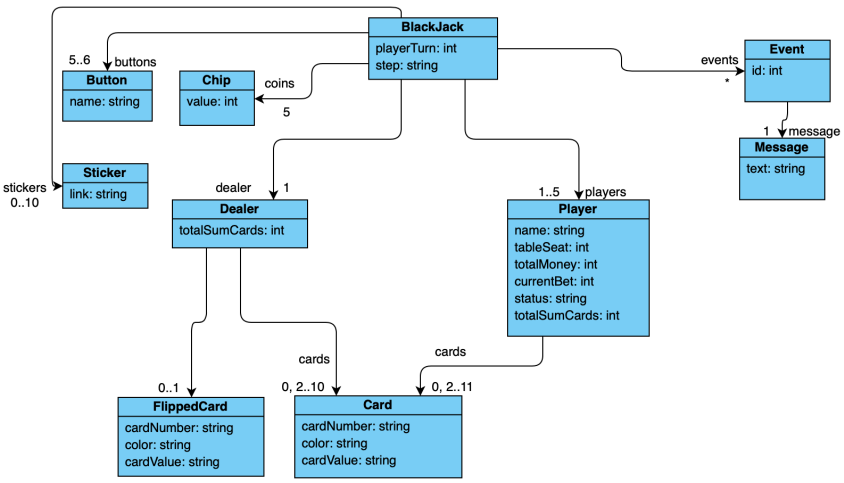
Table 1. Domain model for component View

Screen Type	*Information*	*Interaction*	*Request*
			lobby()
lobbyScreen		goToAbout goToRules goToInitPlayer	rules() about() initPlayers()
aboutScreen		goToLobby	lobby()
rulesScreen		goToLobby	lobby()
initPlayerScreen		goToLounge	handleInit()
loungeScreen	playerName	goToLobby	join(gameID, playerName, balance)

Screen Type	*Information*	*Interaction*	*Request*
	playerBalance playerPoints playerId	newGame randomGame joinGame	newGame() random(playerName, balance) menu(playerID)
blackjackScreen	See UML class diagram below	hit stand double surrender bet(1, 5, 10, 25, 50) clear deal exit newRound	hit(gameID) stand(gameID) double(gameID) surrender(gameID) bet(gameID, betAmount) clear(gameID) nextBet(gameID) exitGame(gameID, playerID) newRound(gameID)
endGameScreen		goToLounge goToLobby	returnToLounge(gameID, playerID) returnToLobby(gameID, playerID)

Following UML Class diagram shows the objects and their attributes that the View component needs to show the correct state of the game (blackjackScreen) to each particular player.

Figure 8. Class Diagram for the View Component



We decided to include the Chip and Button classes to the diagram as well. They do not change their own values during the game and they do not directly indicate the state of the game. However, they are necessary for the player to interact (make bets or perform actions) and this implies changes during the game. Thus, they indirectly affect the state of the game.

Stickers are the animation that are shown when a player wins, loses, surrenders or gets a blackjack.

Implementing the Component View

All of the files that belong to the View component are contained in the folder named **static**. View component contains the **casino.dtd** file that defines the structure of the elements in the **casino.xml** file. DTD also defines which attributes and elements can be used in the document. The **casino.dtd** can be found in **/static/DTD/casino.dtd**. For implementing the View, we used 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, SVGs, plain text or XSL Formatting Objects, which may subsequently be converted to other formats, such as PDF, PostScript and PNG. In **static/XSL**, one can find 4 files with **.xsl** extension. **blackjack.xsl**, **endGame.xsl** and **lounge.xsl** are responsible for transforming the XML document to SVG and the screens that can be seen in **Figure "Game Table"**. In these files, we used global variables which are stored in **bj_global_variables.xsl** and can be linked through the `<xsl:include>` keyword.

For transforming the XML document into an SVG , we used `<xsl:apply-templates>` and select to specify in which order the child elements are to be processed.

Graphic elements were created with SVG and combined with XSLT. Thus, we could control which elements to show and which elements to hide. While implementing this part, we made sure to parameterize the SVG files and to make them reusable and responsive. A file named **bj_global_variables.xsl** was created to contain all global variables of our three XSL files. By means of these global variables it was possible to change how elements were displayed without modifying all the values of that element. Giving SVG properties such as `height, width`, and `aspect ratio` is the first step to getting it to scale. We used the `view box` attribute for this. The view box defines how all the lengths and coordinates used inside the SVG should be scaled to fit the total space available and it defines the aspect ratio of the image. SVG `<defs>` elements were also used to embed definitions that can be reused inside the SVG file. Since they can not be directly displayed, we referenced them by a `<use>` element which specifies where to show the reused shapes via its `x` and `y` attributes. Different gradients, shadows and other filters were defined also and reused in many elements such as buttons or chips.

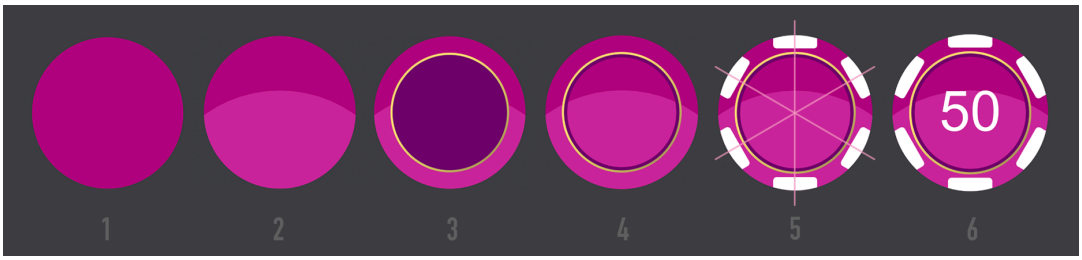
Below we see the interface during a game. Achieving this result took us a lot of time brainstorming and then prototyping. The background is a simple fill color. Moving to the table, it is basically composed of two circles with different radius values. The interior circle has a gradient fill. In order to create the printed label on the table we created that shape in Adobe Illustrator and then we exported it as an SVG file. The text on the label was assigned a path created also in Adobe Illustrator to make it bend. Players and bet zones were created with simple circles with a stroke. Using XSLT we displayed cards according to a rule so that they always stay centered to the player zone. For that, we had to take in consideration the amount of cards a player has. The arrow pointing to each active player is a text element (symbol). Text zones for messages or for bets were created with rectangles with rounded corners. For creating the chips a more detailed description of the process can be found in the next paragraph. Similar techniques used for designing the chips were used also for designing the other buttons.

Figure 9. Game Table

Chips

All five chips were created in the same way. Just values and colors differ from one chip to another. In the following example we will take a look at the steps we performed for creating the chip 50.

Figure 10. Creating a Chip

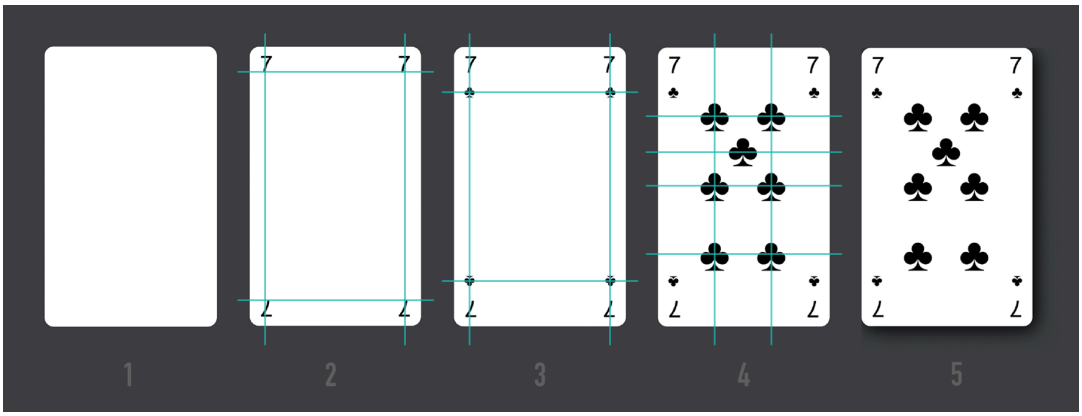


The chip is composed mainly of circles. All elements are positioned according to the center of the chip. For this we used a global variable called `chipCenter`. In the first step we created a simple circle with a fill color. Next we duplicated the same circle but with a lighter color and used a clip-path that we named `cut-off-bottom` to subtract a section from the original shape. This gave us a fake 3D look that we opted for. In the third step we added another colored circle with a golden stroke. We used a linear gradient that we called `goldGradient`. The fourth step is the same as the second one but with a smaller circle radius. In the fifth step we used the same technique as in the second step. We created a white circle as big as the chip and used the clip-path `chip-white-rectangles` to subtract the rectangles from the circle. The six rectangles were perfectly positioned around the circle since we rotated each one 60 degrees according to the center of the chip (see figure above). Finally, we added a text displaying the value of the chip and adjusted its attributes like text-anchor and alignment-baseline to make it perfectly centered inside the chip.

Cards

During the prototyping phase, we opted for a minimalist design for the cards. A card is composed of a white rectangle with rounded edges as shown in step 1. Next we added the value of the card in each corner. The values in the bottom are rotated 180 degrees. Same for the type of the card, 4 symbols are displayed next to the values. Moving to the fourth step, vertical and horizontal lines forming a grid were created as global variables. Depending on the card value, symbols are shown according to the grid. In this example we used four horizontal lines and 2 vertical lines. Finally, a drop shadow is applied on the whole card using a filter called `f1`. This drop shadow creates a depth effect that enhances the visual experience of players.

Figure 11. Creating a Card



Modelling the Component Controller

Controller connects the View and the Modell between each other. This component is stateless. First task of the Controller is to call a method in Model that is related to the View's requests. Second task of the Controller is to respond to the View's request and to define which screen with which information the View has to show next.

Implementing the Component Controller

Requests that may come from the View are indicated in the last column of the table in the section "Modeling the component View". Controller has to guide and handle them and that is why Component has a XQuery methods with RestXQ annotations for each possible request. However, not in all cases the Controller needs to call the Model to respond to the request of the View. Controller can just return a static file for the functions like `lobby, rules, about` and `initPlayer`. The Controller component is also responsible for sending notifications whenever an update in the model occured. In our game the `draw` function is called by a `gameID`, then this function sends the update to all players subscribed via a Websocket. `ShowGames` also functions this way, but sends information only whenever an update in the lounge occurs.

The functions that are not directly called by the View are the helper functions that are called within the Controller. When the Controller needs to call a Model, it uses a request data to and calls the corresponding function in Model. Afterwards it gets the data from the Model's response and creates a response for the View (information and type of screen that View has to show next).

Implementing WebSocket

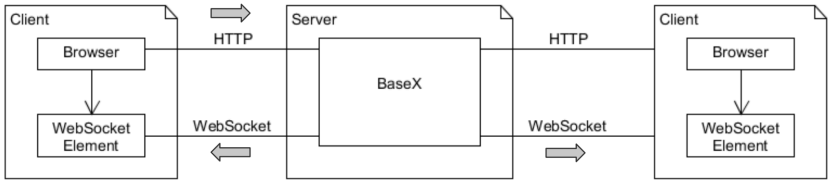
To have a possibility to use a WebSocket Element, there is a need to have a JavaScript library on the client's side. Necessary WebSocket files **jquery-3.2.1.min.js**, **stomp.js** and **ws-element.js** are added to the JS folder that is situated in the static package. WebSocket Element connects to the Websocket Server and builds a bidirectional communication between them.

Controller component file **blackjack-controller.xqm** contains a script for loading the JQuery, STOMP and WebSocket Element.

The file **blackjack-ws.xqm** from the Controller component contains the functions that handle the requests from the WebSocket element from the client side. Whenever a new client connects and subscribes to a websocket connection the `numberOfUsers` in **casino.xml** is updated and incremented. This element is used later on for generating the `ID` of the player.

Whenever a client subscribes to our game , its `ID` and applications `ID` will be mapped to his WebSocket connection using the `websocket:set()` function. This helps to figure out the client `id` based on `websocket:id()`.

Figure 12. WebSocket as a Response Channel [2]



The functions that use the WS Element:

- `wsInit`: this function creates a websocket element for each player and makes him subscribe to /**webSbj/{his ID}**
- `draw`: this function sends an update of a specific game to all its players via `websocket:send()` function
- `showGames`: this function sends an update of the lounge to all players in the lounge via `websocket:send()` function

For the `draw` and `showGames` functions, we used `xslt:transform()` function which allows us to map our XML file to SVG and send that SVG back to the client.

Furthermore, we used the `map()` function, which allows us to map to variables in the XSLT. This `map()` function helped us figure out who is seeing that SVG. For example, if the player that sees SVG is the `currentPlayer` then he is allowed to hit.

Group Prefix

Another requirement was to add our group name as a prefix to the following names:

- To the folder with our project data in the baseX webapp directory.
- To the name of the folder that we will place in BaseX's **static** directory.
- To all paths in RestXQ annotations.
- For all names of databases in BaseX.

However, we decided to put not the **WS** prefix to all of these names, but a **webS**. We did so because we felt that other groups could use the prefix WS to name the functions, pathes and objects related to the WebSocket Element in their code.

III. Project Development

Design Decisions

Through the development of the game we had to make multiple decisions and seek compromises so at the end we had the desired Product.

We decided against implementing the Split function, since split is a rare occurrence in the game. Besides, there is a theoretical possibility that the split occurs multiple times in one round. It would make the modelling and implementation way complicated.

It is mathematically proved that the only case when it really makes sense for the player to do insurance is to guarantee a win in case of the player's blackjack [1]. Since our casino is generous and the player

is guaranteed to win, if he scores 21 points, we decided that it makes no sense to implement the “Insurance” function.

Between an infinite deck and a finite one, we decided for the finite one with only one deck of 52 cards. We purposely chose that because with an infinite deck it's possible to draw the same card multiple times. In addition, a finite deck is more realistic.

We decided to shuffle the deck at the beginning of the game. We also decided not to use a function that randomly chooses a card from the deck. Basically at the beginning of the game the deck should be shuffled and ordered. We did it this way for optimization reasons.

We decided to implement the `id` of the player as an attribute and not as a node because the attribute would be unique and it is better suited for our situation.

After visiting a casino, we decided that 5 players for a table is already enough. Since most tables usually have about 3 players.

In actual blackjack the dealer has a 52% chance to win but we wanted our casino to be more customer friendly. So we decided to give victory to the player in case of a draw.

Problems during Development

- One of the biggest difficulties that we had at the beginning was that two members of our group had problems installing BaseX STOMP. So to solve this we decided to install the game on a remote server that automatically updates the Localhost with the latest modifications from Git. This actually helped us during the testing phase of the game since all the members could join the game together in a matter of seconds. This decision gave us an opportunity to extensively test the game and find bugs faster.

- Converting the game from a single-client-version to a multi-client one was a problem at the beginning. It happened, because our original game model was developed for a single-client application.

- We could not find a way to locally reference pictures, animations and GIFs in the multi-client version, so at first we uploaded the files to a remote server and used HTTP protocol to call them through the internet. Afterwards the Tutor helped us and showed a method to reference the files locally.

- Enabling Websockets in the lounge was a tough task at the beginning, since we were not able to find out which client is currently in the lounge. By the way, after some adjustments we solved the problem.

- The problem with rounding numbers occurred because the user's balance is built on integers. For example, the player leaves the game and enters the lounge with 85.5 on his balance. Then the balance has to be rounded. We decided to round it down, because we create a new user every time when somebody enters the lounge from the lobby. Player can keep his 0.5 balance during the game. Once he leaves, he loses it.

Agile Development, Testing and Possible Improvements

We used a modified and simplified version of **scrums**. In every meeting we made a list of the features that we wanted to implement and of the bugs that we wanted to fix. We discussed how we can achieve our goals and compiled the list as Product Backlog. We used Trello to maintain the backlog list and to follow our progress.

Since we were using a remote server for development, we decided to extensively **test** our game together after every big modification. The test protocol was simple: the four of us joined the game server and played multiple games together trying to break the game. At the same time we documented bugs and emerged problems. We used Trello to document the bugs and keep track of them. Such an approach

also allowed us to control whether bugs were fixed or not. In addition, each team member was engaged in additional testing of the game in his/her spare time.

Following features could be implemented in future releases:

- Split
- Insurance
- An opportunity to delete a game if the game is empty. That option should only be available for the player who started the game (the host of the game).
- Option to be a spectator during the game and never join it (even if there is a free seat).
- The interface can be improved using other dynamic elements.
- Adding background music would probably make the game more enjoyable for the players.

Reflection

Our Application is not just about the basic requirements. Our team's goal was to provide high quality for the gaming experience and it's User Interface. However, the application can still be developed and improved.

Thanks to this course, we gained so much information about the XML-Technologies. We learnt how to build a Webapp and at the same time we had fun. We also received the necessary help from the organizers during the lab. Therefore, taking this course is highly recommended for future students

Since the game is already on the internet, few members in the team are already interested in developing the game further in the future. One idea was to add Google AdSense to the website to generate an income through advertisement. Because of Corona, a lot of people are staying at home bored. So another idea came from here: start a Facebook Ad campaign for the website to bring traffic and at the same time to generate income through Google AdSense as a result.

IV. Tutorials

Installation Guide

There is two possible ways to access the game.

Local Version

1. **Installing the local version of the Game**

   Requirements

   Operating System

   - The game is guaranteed to work on Windows, Linux and Mac OS.
   - The Game functions fully and sometimes partially on some versions of Android or IOS.

   Browser

   - The game works with most browsers, but it's recommended to use Google Chrome since the game has been optimized for it.

   Software

   - **BaseX**: for the single-client-version you can download Basex using this link **http://basex.org**.

- **Basex Stomp**: needed for the multi-client version. Please follow the following Tutorial to install Basex Stomp **http://bstutorial.TheBlackJack.Casino** (Source: XML Praktikum TUM)
- **Eclipse:** we recommend using Eclipse to setup the localhost server.

Game

- You can download the game using the following link **http://Game.TheBlackJack.Casino**

2. **Setting Up the Server**

   After following the Basex Stomp Tutorial, you should copy the game's files to the main directory of BaseX folder shown by Eclipse in its console.

   a. First copy the complete folder webSblackjack from the main directory of of the game in to **[Basex main directory Folder(shown in Eclipse)/webapp/]**

   b. Copy the the second folder webSblackjack found in the game files under **[Game Folder/static/]** directory to **[Basex Folder main directory (shown in Eclipse)/webapp/static/]**

   c. At the end open the following link to setup the server **http://localhost:8984/webSbj/Setup**

3. **Accessing the Game**

   Using the following link the user can access the game: **http://localhost:8984/webSbj/lobby**

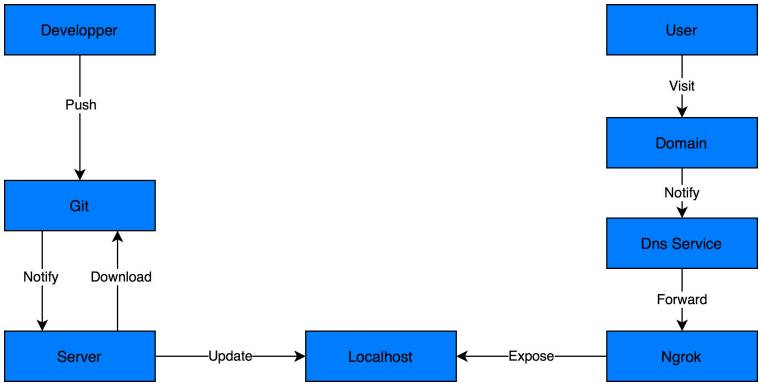
Online Version

The easier way to play the game is to directly visit the website using the following link

www.theblackjack.casino

Server Setup:

Figure 13. Informal Model of the Server Setup



The game is being hosted on a Remote server. Any developer can push his modifications and Git will immediately notify the server that there is a new update available. The server then will automatically pull the new modifications, update the Localhost server and reset it through a new setup.

We are using the service of Ngrok to expose the Localhost to the external networks.

A DNS Service will forward the user after he tries to visit the game to the right Ngrok server and pass on the information of the user.

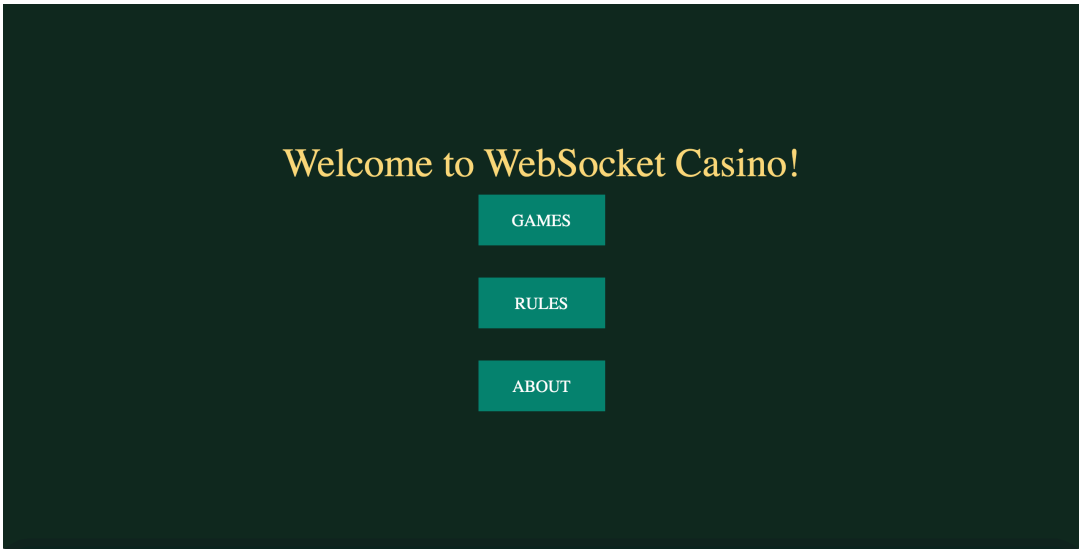
Some modification has been made to the online version in comparison to the local one for performance reasons. As an example the animation, GIFs and Pictures are hosted directly on the remote server and the user accesses them through http service and not the localhost.

Game Tutorial

In the first step, you will enter the url **www.TheBlackJack.Casino** for the online version or using the localhost **http://localhost:8984/webSbj/lobby** for the local version.

Main menu

Figure 14. Main Menu

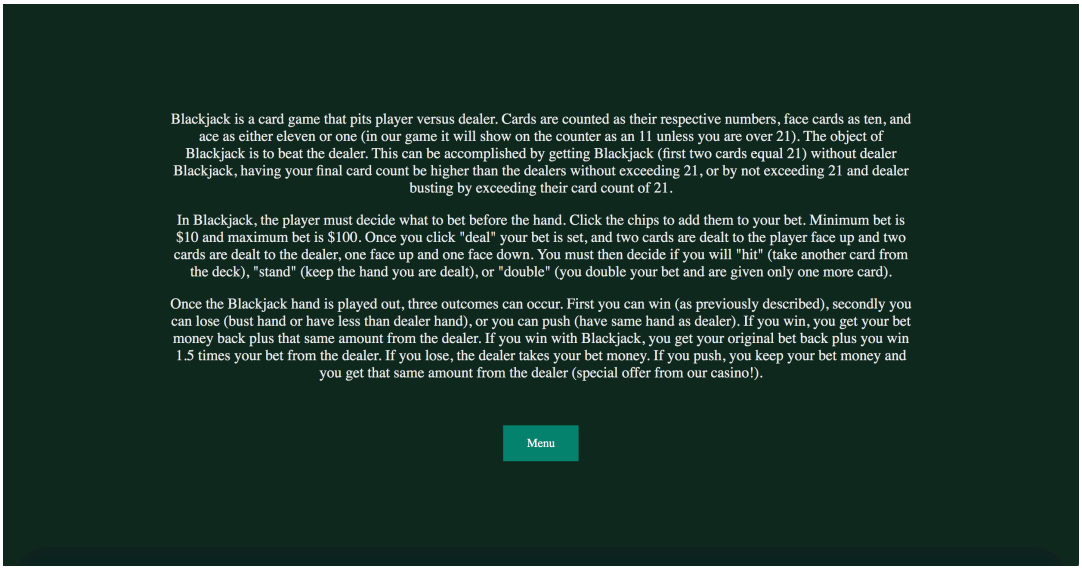


As soon as you start the game, the main menu appears. It shows three tabs named as Games, Rules and About.

- Clicking on Games will open the game for you and guide you to the lounge.
- Clicking on Rules will provide you the rules needed to understand this game.
- Clicking on About shows information about this project.

Rules Section

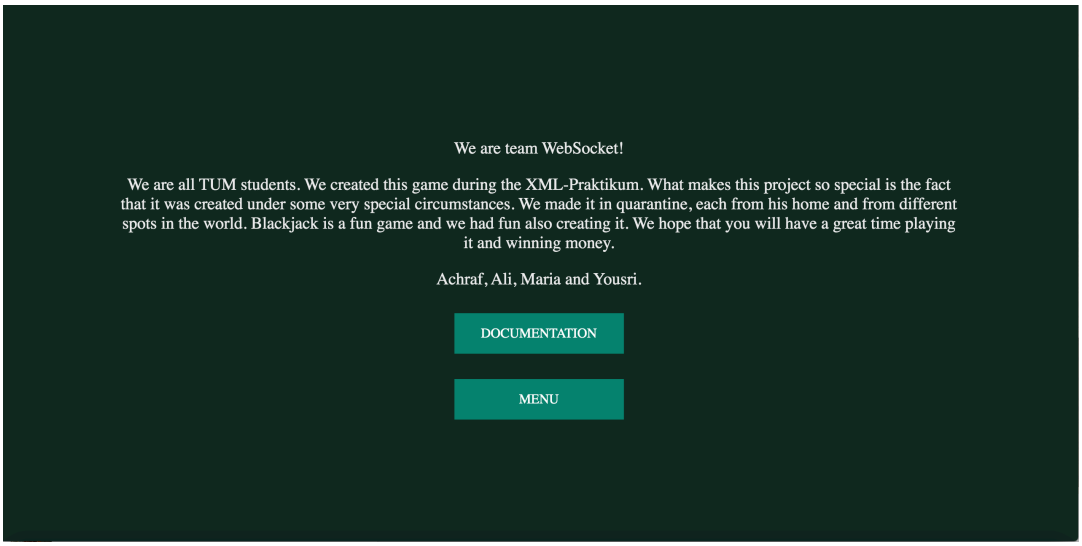
Figure 15. Rules Section



In the Rules section, some BlackJack rules are defined.

About Section

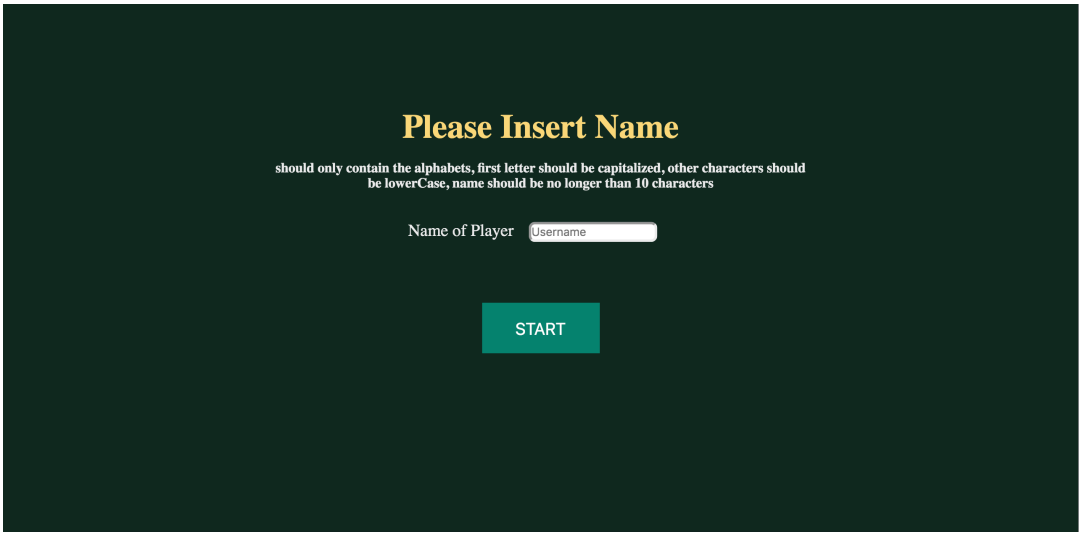
Figure 16. About Section



In the about section, team members are mentioned.

Games Section

Figure 17. Games Section

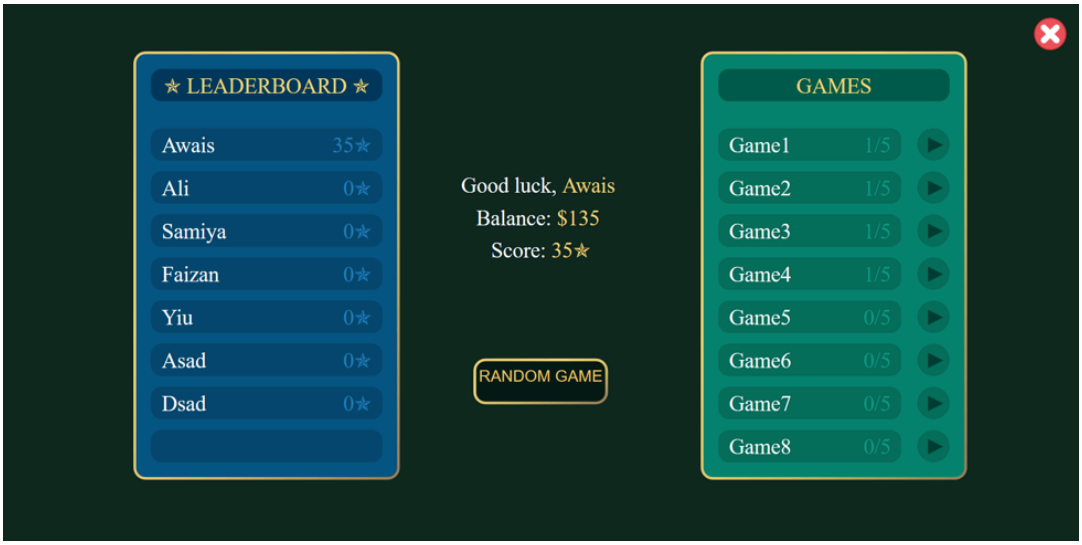


In the following section and to access the game you will have to enter your user name. The user name should start with a capital letter and should not exceed 10 letters.

If the player joins for the first time, then he will get rewarded $100 in his balance. If the player have registered with his username before, then the balance from his last gaming session, will be carried over to the next one. If the player is already registered but his old balance is less than $10 then he will get rewarded a new $100 to his balance since the minimum bet in the game is $10. This feature was added so players get a new chance when they lose all their balance.

Lounge

Figure 18. Lounge



The lounge contains three sections.

- In the **left section** there is the **leaderboard** with a list of the best players in the game. The player having the highest score is at the top of the list. As in the following example the player **Awais** has the highest score and is at the top of it. Before the start of the game you will get $100 in your balance surplus and after winning every game that balance will be added to your score.

Table 2. The formula of the score for the Leaderboard is as the following:

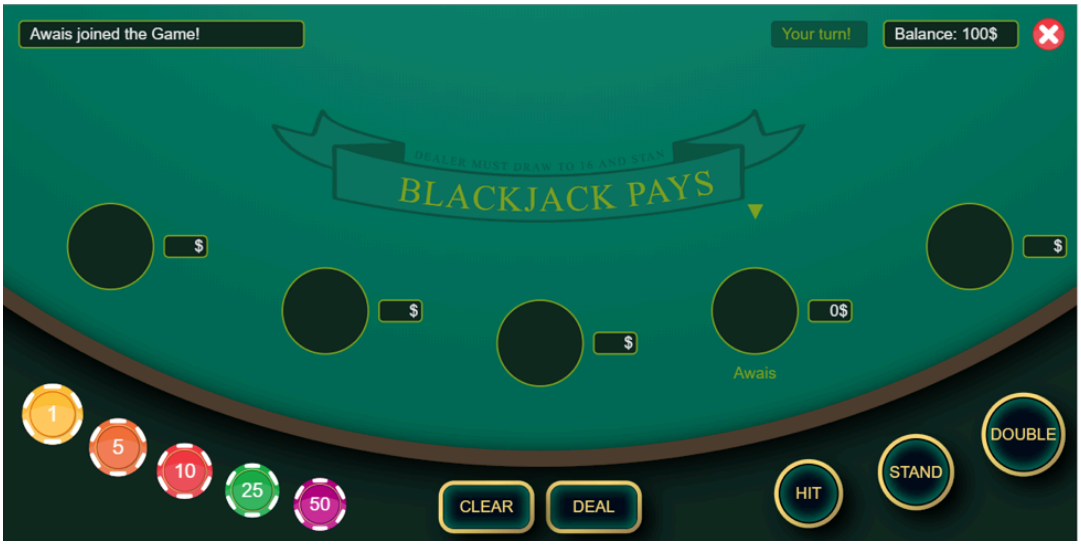
Game scenario	Score Formula
Winning or Drawing	New Score = old score + bet from latest game
Winning with blackjack	New Score = old score + 1.5 * bet from latest game
Losing	New Score = old score - bet from latest game

- In the **middle section** the player's name, his available balance and his total score are shown. The button **New Game** enables the player to start a new game. Only 8 games in total can be played in parallel and if there are 8 available already, then the **NewGame** button will disappear. The button **RandomGame** enables the player to join a random game from the current games available. It will only be visible if there is at least one game available.

- In the **right section**, there is the list of games that the player can join with the number of players playing currently in each game. By clicking the **Play Button** shown as the symbol ▹, the player will join the desired game. If a Player joins a full game with 5 Players, then he will initially just watch the game being played in front of him. As soon as one of the playing players leaves, then he will immediately join the game.

The Game

Figure 19. The Game

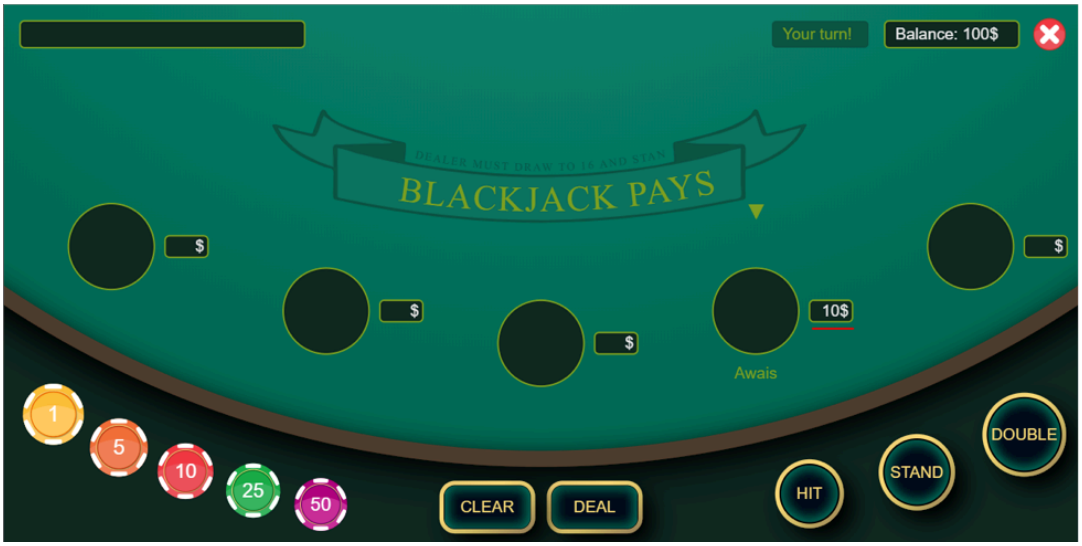


It’s a simple card game and no hard end tricks are needed. You just need to be sharp and lucky, that’s all. You can bet a minimum of $10 or a maximum of $100. Just select your manoeuvre wisely and be clear for what you want to do. In case you win or draw, you will win the betted amount. If you lose then you will lose the betted amount. If you get a blackjack you will win 3/2 of your Bet. It’s an easy card game. You just need to play it often to get experience in it and then you can easily go forward and win big.

Layout

The button on the right side is **Double**, it means doubling the betted amount and adding only one more card to your hand. It's only available if the player has only two cards in his hand and the sum of the hand is less than 11. **Stand** means keep the bet going and try your luck, however **Hit** means adding a card from the line-up. If you click the **Deal** tab, then the round will start. The **Coins** at the right side enable the player to raise the amount of the bet. The **Clear** button enables the player to clear his bet and it is only available before the bet is placed. The **Give Up** button is only available at the beginning of the game and will make the player lose half of his bet. The **New Round** button will start a new Game and only one player has to press it to start a new one. At the top right corner there is an **X** button, this button will enable the player to close the game or to go back to Lounge. Next to it is the **Balance** of the player being shown. At the top left corner there is a feedback **Message Box** that shows the player important information about the state of the game.

Figure 20. The Game



After joining the game, you will add money to your bet and start playing after pressing the deal button and all you are trying is to beat the dealer either:

- By drawing a hand value that is higher or equal to the dealer's hand value and at the same time less than 22 in value
- By drawing a hand value of 21 on your first two cards, when the dealer does not.

Example

In the following example, you got 15 as a hand sum. You can hit to get a new card or stand with your number and let the dealer show his cards.

Figure 21.

In case you stand with your cards, the dealer hits and gets more points than you and you will end up having less than 21 points and lose your bet. Pressing the new round button will get you to a new round.

Figure 22.

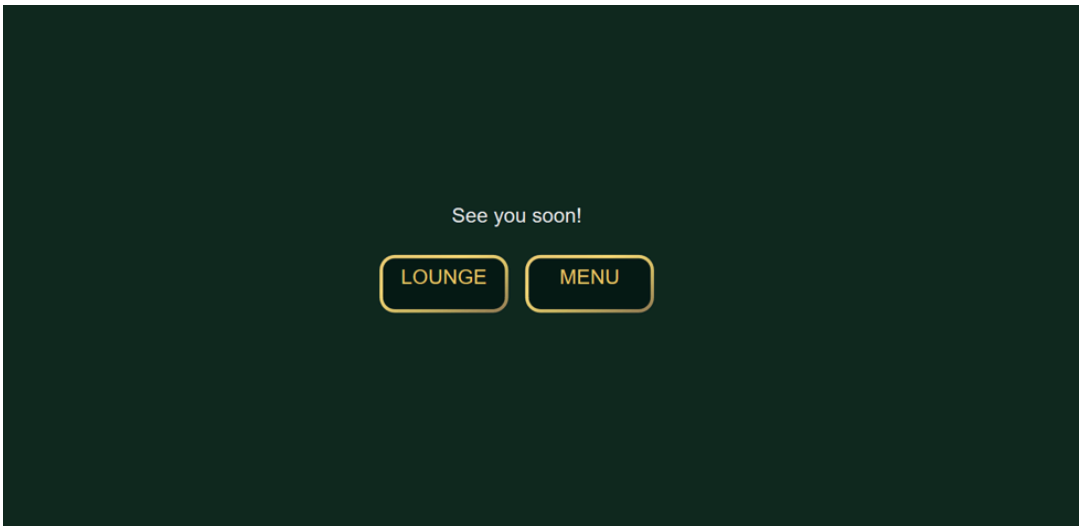
If all goes well, you will win your Bet.

Figure 23.

If you click to close you will get two options, by selecting **MENU** you will go to the step Main Menu. By selecting **LOUNGE** you will go to the lounge, where you can see your score.

Figure 24.

Figure 25. Exit



Troubleshooting

- In case there is a problem with the game please try resetting the game through either **http://localhost:8984/webSbj/Setup** for the local version or **http://setup.TheBlackJack.Casino** for the online version.
- To access the database to be able to debug the game please visit **http://localhost:8984/dba** for the local version or **http://dba.TheBlackJack.Casino** for the online one. For the database username: admin and password: admin.
- If a player's current balance is less than 10$, then he will be kicked out from the ongoing game since he has less money than the required minimum bet. To solve this please quit using the **X** button on the top right corner and go back to the main menu and sign in again using your user name. After it you will be rewarded a new 100$.
- If you want to leave a current Game, then please make sure to use the **X** Button at the top right corner, otherwise the current game may get stuck in your turn. If this happens either reset the server or join a new game.
- If a username is not accepted at the login phase, please check that the username is written in the asked format. Also check that no other open browser tab uses the same username.

- If the game is accessible, but in some way not working, then please try with another browser.
- If the game is not accessible, then please try with another browser. If you can't access the online version, try to access it using the VPN.
- If you are having problems installing Basex STOMP, then you should try changing Central Maven repository in the POM.xml (you will find this files in the main directory of Basex Stomp) to **http://insecure.repo1.maven.org/maven2/** and make sure to replace all the references to **http://repo1.maven.org** or **http://repo.maven.apache.org/** with **http://insecure.repo1.maven.org**.

A. Tables with Functions

Following tables describe all of the functions that were used in the implementation. Description inherits the following format:

1. Name of the function
2. Parameters
3. Explanation
4. Return value

Table A.1. Functions in game.xqm

1. ***getCasino*** 2. –- 3. this function return our Casino 4. the casino model	1. ***getGame*** 2. ***@gameID*** the game Id 3. this function return the Game using the ID and searching it in our casino 4. the whole module of that Game	1. ***createGame*** 2. ***@maxBet*** the maximum Bet of the Game; ***@minBet*** the minimum Bet of the Game 3. this function create a new Game by calling the method createNewGame 4. the newly created Game
1. ***insertNewPlayer*** 2. ***@gameID*** the ID of the game in which the player will be inserted to; ***@playerName*** the name of the User; ***@balance*** the balance of the User; ***@id*** the user's ID; ***@tableSeat*** his Seat on the Game should be between 1 and 5, -1 if he is waiting and watching 3. this function insert the user as a new player using his name 4. update the Game	1. ***join*** 2. ***@gameID*** the ID of the Game to be joined to; ***@playerName*** the name of the player; ***@balance*** the balance of the player 3. this function is called when the user wants to join the game , it checks if there is freeSeat it will insert the player . Otherwise it will add in the Waiting list and give him a -1 as a tableSeat . Finally it will delete the player from the Lobby List 4. update the Game model	1. ***insertGame*** 2. ***@newGame*** the newly created and generated game 3. this function insert the new Game into the casino model 4. update the casino model
1. ***createNewGame*** 2. ***@maxBet*** the maximum Bet of the game; ***@minBet*** the minimum Bet of the game;	1. ***startGame*** 2. ***@gameID*** the ID of the game to start	1. ***checkScores*** 2. ***@gameID*** the ID of the Game

@playerNames the player of names to be added to the game; ***@balance*** the balances of the player 3. this function generate a new Game by creating the players Element and generating the game Deck 4. the generated game model	3. this function is called when cards have to be served, i.e the game has to start. 4. update the game Model by adding cards to players	3. this function checkScores after each round, if there is no player left, then it will change the state of the game or step to gameOver 4. update the game model
1. ***intiliazeSum*** 2. ***@gameID*** the ID of the game 3. this function is called to calculate the Sum of cards before the game starts 4. update the game Model	1. ***dealerTurnHelper*** 2. ***@gameID*** the ID of the game; ***@sum*** the sum if the dealer withdrawal; ***@limit*** how many Cards have to be withdrawn so far 3. this function checks Recursively how many the Dealer has to draw Card in order to reach 17 or higher 4. how many Cards should the dealer withdraw	1. ***dealerTurn*** 2. ***@gameID*** the game ID 3. this function let the dealer withdraw the Cards in order to reach 17 or higher 4. update the game model
1. ***checkWinnings*** 2. ***@gameID*** the ID of the Game 3. this function return our Casino 4. the casino model	1. ***updateEvents*** 2. ***@gameID*** the id of the Game 3. this function generate Event messages to give to each player using his ID 4. update the events of the game	1. ***updateSeats*** 2. ***@gameID*** the ID of the game 3. this function give a waiting Player a seat if there is one available 4. update the game model and it's players
1. ***newRound*** 2. ***@gameID*** the ID of the game 3. this function is called when a user clicks a new round. It change the players variables like currentBet, dealer's and player's cards and update the free seats 4. update the game model	1. ***addUser*** 2. ***@playerName*** the name of the player; ***@balance*** the balance of the player; ***@id*** the ID of the User 3. this function is used to add a new User to the casino. It checks first if the User already exists, if so then a new User will be created with a new ID but with the same amount of money and points. Otherwise a new User with a new ID will be created 4. updates the game model	

Table A.2. Functions in action.xqm

1. ***nextBet***	1. ***bet***	1. ***stand***

2. ***@gameID*** the ID of the game 3. this function returns our Casino 4. the casino model	2. ***@gameID*** the ID of the game; ***@betAmount*** the amount to be added to the current Bet 3. this function is called when The player wants to add a bet amount to his currentBet 4. update the game model	2. ***@gameID*** the ID of the game 3. this function is called when the player wants to stand 4. update the game model
1. ***double*** 2. ***@gameID*** the ID of the game 3. this function is called when the player wants to double, it should check if he has less or more than 11 sum of cards 4. update the game model	1. ***clear*** 2. ***@gameID*** the ID of the Game 3. this function is called when the user clicks clear and want to remove his bet and make a new bet 4. update the game model	1. ***hit*** 2. ***@gameID*** the ID of the Game 3. this function is called when the player wants to hit 4. update the game model
1. ***surrender*** 2. ***@gameID*** the ID of the game 3. this function is called when the player will surrender, only if he has 2 cards 4. update the game model	1. ***exitGame*** 2. ***@gameID*** the ID of the game; ***@playerID*** the ID of the player to exit 3. this function is called when the player wish to exit the game, it should check which state the game is in 4. update the game model	

Table A.3. Functions in player.xqm

1. ***deletePlayer*** 2. ***@gameID*** the ID of the game, in which the player will be deleted 3. this function deletes player from a game 4. update the game model	1. ***insertPlayer*** 2. ***@gameID*** the ID of game, in which the player will be inserted; ***@player*** the player to be inserted; ***@tableSeat*** the seat of the player to be inserted 3. this function inserts the player to the game and based on his tableSeat he will be placed in a certain position 4. update the model by inserting the player	1. ***new Player*** 2. ***@name*** the name of the player to be inserted; ***@balance*** the balance of the player to be inserted; ***@id*** the ID of the user based on name; ***@tableSeat*** the player's seat 3. this function created a new Player based on his properties 4. a player node
1. ***new User*** 2. ***@name*** the name of the User; ***@balance*** the balance of the new user; ***@points*** the points of the new User; ***@id*** the id of The new User		

3. this function allows you to create a new User based on his properties 4. the new User		

Table A.4. Functions in cards.xqm

1. ***shuffleCards*** 2. -- 3. this function shuffle Cards and return them 4. the shuffeled cards	1. ***calculateDealerValues*** 2. ***@game*** the Game; ***@player*** in this case is the Dealer; ***@limit*** the number of cards to be withdrawn from Deck 3. this function calculate the amount of the dealer Cards plus the first $limit cards from our deck 4. the number of Cards	1. ***calculateCurrentCardValue*** 2. ***@game*** the game in which the player is; ***@player*** the player; ***@cardValue*** the value of card to be added to deck of player; 3. this function calculate the total sum of cards of the player , this function is called for example in hit, stand etc. 4. the new total sum of cards of the player
1. ***getWholeDeck*** 2. -- 3. getter function for Deck 4. the whole deck of our casino		

Table A.5. Functions in helper.xqm

1. ***random*** 2. ***@number*** integer 3. creating random number 4. random number	1. ***randomNumber*** 2. ***@range*** integer 3. this function uses Java function until generate-random-number and is generally available 4. a random number in [1, $range]	1. ***generateID*** 2. -- 3. creates ID 4. ID

Bibliography

[Elektronisches Publizieren: Document Engineering im World-Wide Web] Philipp Ulrich, 19.12. 2019.

[[2] Multi-Client Webanwendungen mit XML Technologien] Prof. Dr. Anne Brüggemann-Klein. Dipl.-Inf. Univ. Marouane Sayih.

[WebSocket Element] Philipp Ulrich, 06.06.2019.

[XQuery und XQuery Update Facility im XStack] Anne Brüggemann-Klein. Philipp Ulrich .

[X Stack Demo GN Version 2.0] Anne Brüggemann-Klein.

[[1] Blackjack Insurance Explained] https://www.onlinegambling.com/casino/blackjack/insurance-bet/.