*SOFTWARE DESIGN SPECIFICATION*

*Team: Online Casino/SportsBook*

*Team Members:*

*Keaton Dinger*

*LiLou DeRivera*

*Nathan Sikon*

*Hitesh Kukreja*

**1.0 Introduction**

This section provides an overview of the entire design document. This document describes all data, architectural, interface and component-level design for the software.

**1.1 Goals and objectives**

Create a smooth-running blackjack game using the Python coding language, then use HTML to develop the frontend of the game.

**1.2 Statement of scope**

The inputs will require different calls and plays from the player that a standard game of blackjack has; bet (ante), hit, stay, double down, split. The back end handles game logic, while front end handles the UI as the user interacts with the game **1.3 Software context**

Blackjack is relevant in the entertainment gaming industry; it is one of the most well-known games for gamblers and recreational players alike. A software such as this can provide casual to high stakes gambling in an online setting.

In the case of gambling with real money, the software would have to comply with gambling laws wherever the software is run.

**1.4 Major constraints**

In a real-life setting, the software would have to comply with online gambling laws where it is launched. Some places may not allow such software, so blocks would have to be made according to location.

**2.0 Data design**

A description of all data structures and databases.

**2.1 Data structures**

The major portions of data used will involve game logic data, this includes card values, player(s) information (name, money/chip count, current hand value), dealer information (hand value), cards left in the deck.

**2.2 Database description**

For the player table, the stored data includes: (player\_id, name, balance,).

For the bets table, the store values include: (bet\_amount, game\_result, winnings).

**3.0 Architectural and component-level design**

A description of the software architecture is presented.

**3.1 Architecture diagrams**

Various views (logical, process, physical, development) of architecture are presented with descriptions.

A diagram of a process

AI-generated content may be incorrect.

* Frontend:
  + Displays UI, sends player actions, and receives game updates
* Backend:
  + Processes game logic, handles player/game actions, and manages database
* Database:
  + Stores player and game information

A diagram of a process

AI-generated content may be incorrect.

- The player interacts with the front end of the software (HTML).

- From here, the player interacts with the UI to play the game and make their calls

- The choices made on the front end are sent to the backend, where game logic is processed

- The logic changes are sent to the frontend to make visual changes and open up new options

- New options are then presented to the player with changed logic

A screen shot of a game

AI-generated content may be incorrect.

- The graph illustrates what features need to be completed first within each incremental phase before the next phase is reached.

**3.2 Description for Components**

A description of major software components contained within the architecture is presented. Section 3.2.1 is repeated for each of n components.

**3.2.1 Component n description**

**3.2.1.1 Interface description**

**FRONT END**

Inputs:

* User actions, game state data from backend

Outputs:

* Visual representation of game state, event notifications to backend

Exceptions:

* Network connection failures, Invalid game state responses

**BACK END**

Inputs:

* Player action requests, player identification data

Outputs:

* Updated game state, game result information

Exceptions:

- Invalud player actions, game state persistence failures

**3.2.3.2 Static models**

A diagram of a user flow

AI-generated content may be incorrect.

**3.2.3.3 Dynamic models**

A diagram of a game

AI-generated content may be incorrect.

**3.3 External Interface Description**

The software's interface(s) to the outside world (other software or hardware systems) are described.

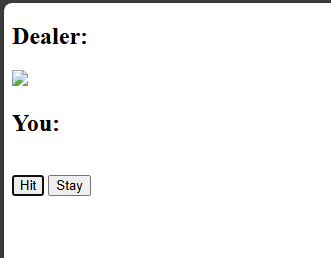
* **User Web Interface**: Backend: HTTPS, while front end is undecided but could be react.js with WebSocket
* Password and usrrname via HTTPS POST
* Game Results: { “result:” “win”, “payout” : 200 }
* Base URL: https://[domain]/api
* Key Endpoints: POST /auth/register, POST /game/blackjack
* **Browser Compatibility**: Chrome, Firefox, Edge, etc.
* **User Data Flow:** Browser --> HTTPS ---.> application server
* Application server ---> websocket ---> browser

**4.0 User interface design**

A description of the user interface design of the software is presented.

**4.1 Description of the user interface**

A description of user interface including screen images or prototype is presented.



* Homepage: A start screen that welcomes the user, shows account, currency, and start button
* Game Lobby: Shows cards with pictures, the poker table, chips, and the options to play your cards
* Dashboard: User balance, bets, and current cards with dealer and user

**4.2 Interface design rules**

Conventions and standards used for designing/implementing the user interface are stated.

* Responsive Design: Website must feel state of the art and have fast response time when users are interacting
* Color Scheme: Must feel like they are playing on a green board like you are at a poker table
* Fairness: Provide a fair and intuitive algorithm for game outcomes

**5.0 Restrictions, limitations, and constraints**

Special design issues which impact the design or implementation of the software are noted here.

* Legal: Compliance with state/government gambling laws in specific regions
* Technical: Limited to web platforms (no application for desktop or mobile devices)
* Performance: Game must handle concurrent users without lag and issues
* Security: Protect users' data and currency. Detect cheating if happening and ban users for it.
* Fairness: Provide fair algorithms for game outcomes

**6.0 Appendices**

Presents information that supplements the design specification.

**6.1 Requirements traceability matrix**

A matrix that traces stated components and data structures to software requirements is developed.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Requirement ID | Component | Design Component | Status | Requirement Type | Test Case ID |
| REQ-1 | User Authentication | User Management 3.2.1 | In Progress | Functional | TC-101 |
| REQ-2 | Simulated Betting System | Betting Engine 3.2.1 | In Progress | Functional | TC-201: Verify bet validation |
| REQ-3 | Blackjack Simulation Logic | Game Logic 3.2.1 | Complete | Non-functional | TC-301: Verify by playtesting |
| REQ-4 | Virtual Currency Management | Virtual Wallet (3.2.2) | In Progress | Functional | TC-401: Verify currency updates |
| REQ-5 | Game outcomes must be provably fair | Game Logic 2.2 | Complete | Non-Functional | TC-501: Verify game RNG seed. |

**6.2 Implementation issues**

* Fake Currency Integrity: Players may find ways to manipulate the virtual balance (e.g. bugs, exploits, hacks)
* Fairness and Trust in Simulation: Users might think the game is rigged if the randomness odds are poorly implemented
* Legal Gray Area: Some jurisdictions may classify the simulated gambling/blackjack as real and not allow for playability
* Privacy: Make sure website is safely implemented and secure with HTTPS to prevent a unsecure website and allow users to be vulnerable to hackers