Zimuzo Ezeozue's Crossover Project Evaluation
zimuzostanley@gmail.com
+2348110569529

Design Document for a Real-time Auctioning Platform

Purpose

This project builds a critical feature in a multiplayer game platform. It enables players perform the following:

1. Place game tokens: *bread, carrots or diamonds* (one type at a time) up for auctioning and set the starting bids.
2. Bid for other players' tokens.

The algorithm for resolving bids is as follows:

1. After a 90 second duration, as stipulated from the requirements, if there have been no bids, the auction ends without any external effect if no player has bidded
2. If a bid arrives within the last 10 seconds of any auction, the time left shall be incremented to 10 seconds.
3. There can be only one active auction at a time. Any new auctions will wait in a queue.
4. The following happens when an auction ends:
    a. The tokens go to the highest bidder while the coins used in purchasing are deducted from his/her account.
    b. The seller's inventory and coins also reflect the appropriate gain and loss in coins and tokens respectively.

High Level Data Model

The Data Model consists of 3 entities: ***Inventory, Player*** and ***Auction***. Their names are intuitive in the context of the project.

- **Inventory:** Holds the token stock of Players.
- **Player:** Holds a player's personal information including their *coins.*
- **Auction**: Represents a FIFO queue data structure.

Zimuzo Ezeozue's Crossover Project Evaluation
zimuzostanley@gmail.com
+2348110569529

Relationships

1. There is a One-to-One mapping between the Inventory and the Player.
2. There is a Many-to-Many mapping between the Player and the Auction

A diagram is attached below.

Auctioning algorithm

1. When the server is started, the oldest uncompleted Auction is instantiated as a singleton object, ready to accept requests from a player, or expire when its time runs out.
2. If the server dies or restarts in the middle of an auction the pending auction would be reset to a pristine state for continuation on the next server start.
3. This object starts a countdown timer for the duration of the auction.
4. When this object expires, it is flagged as ***done*** in the db.
5. The singleton is destroyed along with its resources, including its timer.
6. There's a 10 second wait to display the winning bids to all the users. The loop from **#2** -- **#4** is then repeated.
7. The *new oldest queued* object is instantiated (if there is any) and the loop continues until there are no more *queued* auctions in the database.

The program is event-driven: the state of the singleton auction changes on actions from the players or its timer. Communication between the server and clients is facilitated by WebSockets. The client (browser) has it's own timer serving as a visual cue of the the auction progress to the players.

Keeping the server and client timers in sync over the network is not perfect, but the accuracy may suffice for a first phase.

Technologies Used

HTTP server: Node.js with the express framework

Database server: MySQL

WebSocket library: Socket.io

Frontend Javascript framework: AngularJS

Frontend UI Framework: Bootstrap

Zimuzo Ezeozue's Crossover Project Evaluation
zimuzostanley@gmail.com
+2348110569529

Backend Unit Test runner: Mocha

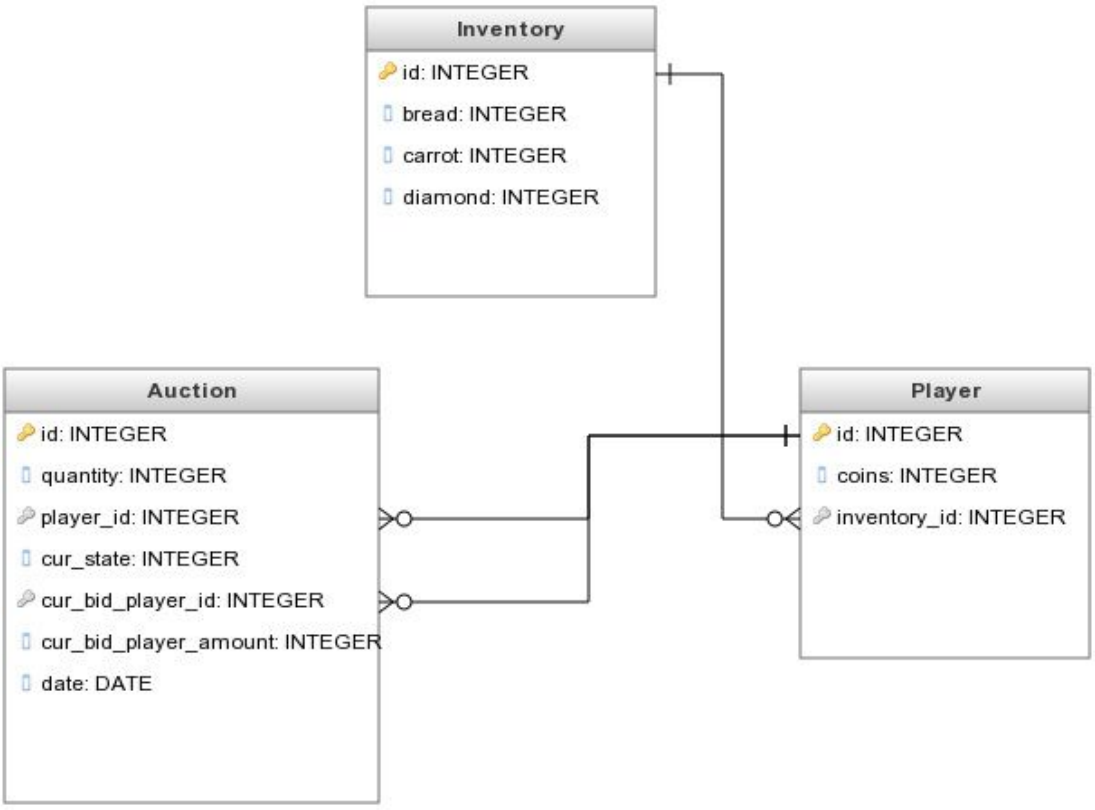
Frontend Unit Test runner: Karma/Mocha

Frontend End to End Test runner: Protractor/Jasmine

Backend Package manager: NPM

Frontend Package manager: Bower

Entity relationship diagram*



*Please disregard the cardinality of the relationships. The tool I used to draw it limited further manipulation to a premium plan.