SYSTEM DESIGN DOCUMENT

1. Introduction

This System Design Document has been created to outline the proposed system design for back end applicaton for auction.

2. Purpose

The purpose of this System Design Document is to provide a description for how this auction application will be constructed. The Systems Design Document was created to ensure that the app design meets the requirements specified in the project requirements documentation. The System Design Document provides a description of the system architecture, database design, WebApi routes, Windows Services and security.

3. Design overview

The application will consist of 3 main interdependent units :

- Database;

- WebAPI service;

- Windows service.

The database is responsible for storing data about the components of the auction.

The WebAPI service is used to interact the user with the application, and to fulfill the main tasks of the program.

The Windows service is responsible for sending e-mails to the user.

The user must be able to perform such operations (These operations are described in project requirements documentation) :

1. Register and Login (simple email/password auth, without email confirmation).

2. Look through the lots in the auction.

3. Create/delete/update their own lots (update and delete options only available when there are no bids on the lot).

4. Bid on other user lots.

5. Create subscriptions for lots they placed a bid on.

6. Receive emails about:

   a) Someone placed a bid higher than his/her (the email should contain information about that bid).

   b) Auction on their lot finished (the email should contain detailed information about bids history, final price, and information about the person who won the lot).

   c) When your bid won the lot.

7. Get a history of the bids on their lot.

8. Each user should have a balance on their account and the ability to add funds to it and receive funds from it (no need to involve payment systems, just add the amount of imaginary money to the imaginary balance). When the user places a bid - money from his account should be frozen. When someone places a higher bid - unfrozen. When he won a lot - the balance amount should be decreased.

4. Database design

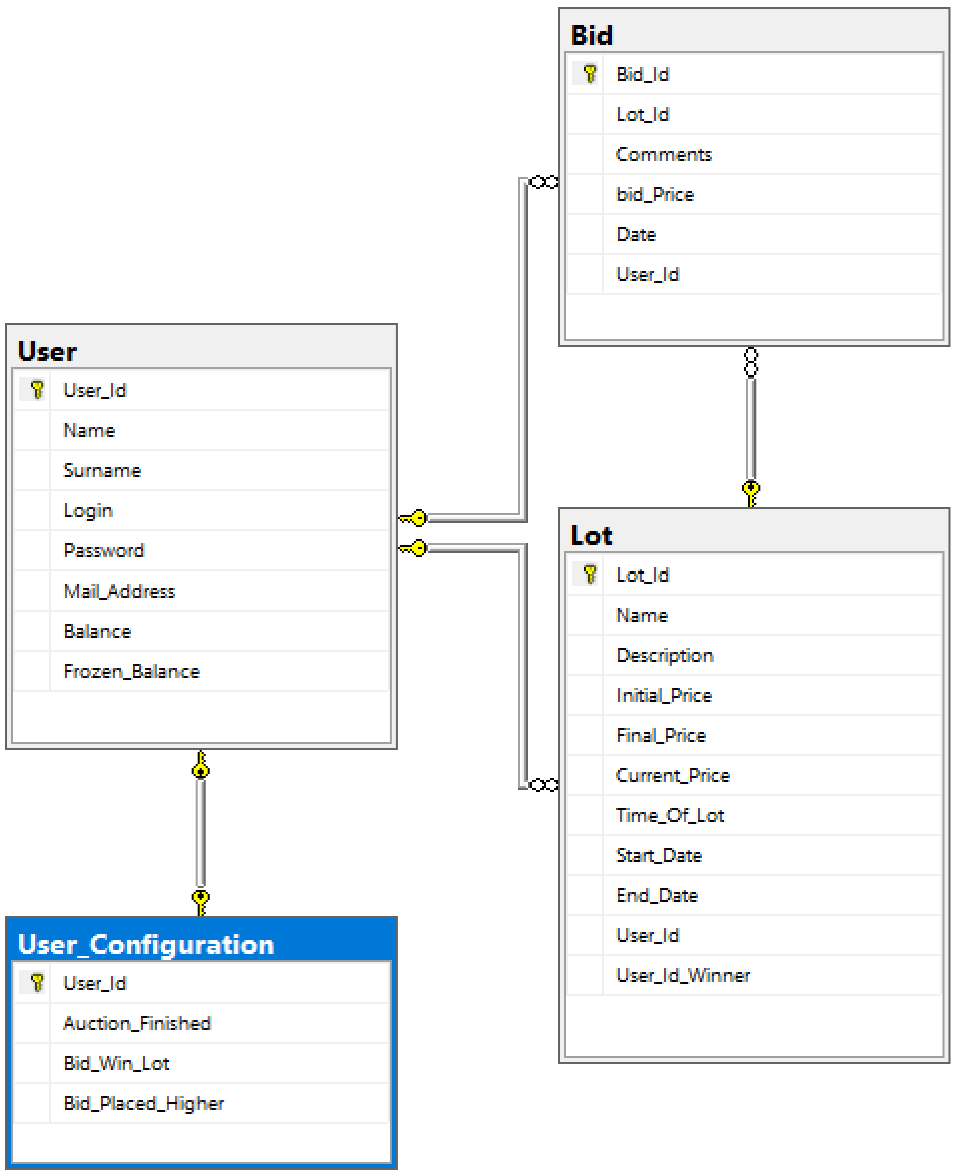
This paragraph describes the structure of the database, the database is the main application data repository.

Picture 1. describe main database diagram. Database of app consist of 4 tables : User , Bid , Lot , User_Subcription .

1. User table – store information about user : credentials , information , balance information. Table has 3 connection with tables : Bid (bets that the user did), Lot (the lots that user has placed), User_Configuration;
2. Bid table - store information about user’s bids : date of the bet , amount . Table has 2 connection with tables : User(All bets that the user has made), Lot(the bets relate to some lot);
3. Lot table – store information about published lot : name (can contain name of the published product), InitialPrice (the price

that was put up for publication), FinalPrice(the final price at which the lot was closed), TimeOfLot StartDate EndDate (date of publishing and closing), User_Id (власник лоту) , Lot_Id_Winner (id of winner). Table has 2 connection with tables : Bid (Lot must store history of bets) , User(Lot is published by some user).

4. User_Configuration – The service proposes to send letters in the following cases: 1- when the lot is expired - notify the owner that the lot is expired and the user who has made the highest bid that the user has won the lot, 2 - when the user places a bet, notify the user whose bid is now lower. This table will store user configurations that specify which emails to send and which not to send.



Database Diagram

5. WebAPI detailed design

All WebAPI routes are described in swager.yaml file provided with this document.

In general:

1. To have a possibility to perform needed CRUD operations with Lots , we have **lot** controller that implements next methods:

   api/lot (GET) – to get lots of other users ,

   api/lot/{id} (GET) – to get lot by Lot_Id ,

   api/lot (POST) – to create new lot ,

   api/lot/{Lot_Id} (DELETE) – to delete lot by id ,

   api/lot/{Lot_Id} (PUT) – to update lot by id.

2. To have a possibility to perform operations with users : refill the account , login , logout , change configurations we have **user** controller with methods :

   api/user/login (POST) – logs user into system ,

   api/user/logout (GET)– logs out ,

   api/user/account (GET , POST) – withdraw from the account , add to account ,

   api/user/account/info (GET) – get account information.

   api/user/configuration (GET , POST , PUT) - get information about user congigurations (* default configurations (set all flags to false) are installed when the user successfully registers).

3. To have a possibility to create a bet , get bids of some lot (lot history) , we have **bid** controller with methods :

   api/bid (POST) – create new bid on the user lot ,

   api/bid (GET)– get all bids of some lot.

6. Windows service detailed design

Windows Services are applications that typically start when the computer is booted and run quietly in the background until it is shut down.

We need a service to receive emails, emails will come in such cases:

1) Someone placed a bid higher than his/her (the email should contain information about that bid).
2) Auction on their lot finished (the email should contain detailed information about bids history, final price, and information about the person who won the lot).
3) When your bid won the lot.

SmtpClient client is used to send the email , this allows you to create emails to fill them with information and send them to a specific email address.

For these tasks we need 2 services : **SimpleService** , **AuctionEmailSenderDemo.**

Simple Service: this service will send an email if someone makes a new bid, the email is sent to the user whose bid is covered. For this purpose, when the user makes a new bid text is formed with information about that bid, this information is recorded in a text document, the service will monitor changes in the text document, if any changes have occurred in the document, it will retrieve information and send a message to the user.

AuctionEmailSenderDemo: this service will send information to the owner of the lot and the winner of the lot when the lot is expired, as well as set the final price and the winner identifier in the database. For this, the service will connect to the base every 5 minutes and will check the current time with the end time of all lots.

7. System Security

For user authentication web service use bearer token authentication method.

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information

between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair
using RSA or ECDSA.

Bearer token authentication method implies that in order to access any data, the user must obtain an access key (token). In order to receive token, the user must send the necessary data (username and password), which he specified at registration. Obtaining token is done in **api/user/login** route (user must specify username , password and grant_type in body of request) , in response user will get token.