Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 19 Bratislava

Software modeling

Charity e-auctions

Program documentation

Matej Hrnčiar
3. ročník, 5. semester
Utorok, 14:00

The final product of software modeling should ideally be a working software, however, model not always correctly describes the software and some parts may need to be modified for the software to actually work. This program is demonstration how model of the software needs to be changed in code to work better. Not everything from the model in Enterprise Architect has been implemented and during programming, some methods and procedures have been greatly modified. The reason is, that during modeling, methods were added to classes that were logically related to it, but in program, it was better to move them to other class, for example *makeBid()* method in model is in *Bidder* class, but in the program, its contents have been moved to *Bidding* class (mind the difference: *Bidder* is a type of user that participates in *Bidding*), where it can directly modify the value of the highest bid and save the ID of highest bidder. These modifications will be mentioned throughout the documentation.

Program comes with graphical user interface, which should be ideally separated from program logic and accessed through *Observer* or other design patterns, but for the sake of demonstration of basic functionality, it is fine as it is - in the same class as the program logic.

Following classes have been implemented: *User* and child classes *Bidder, Organizer* and *Operator, Auction, Bidding, Charity* and *Datetime*. New classes *Main* and *Dashboard* have been added.

Another difference from UML model is database, which was modelled to be PostgreSQL database, however setting up and establishing connection to such database only for demonstration purposes is complicated, so the database has been replaced with SQLite, which is whole saved in one file "*charity.sqlite*". Further instructions to set up connection to database are written in README file in program folder.

Project has been written in Java 17, but it should be backwards compatible.

Main

The purpose of Main class is to establish connection to database and generate Dashboard class, which displays the welcome screen with sign in and sign up options, trending auctions and other information.

Variables:

None

Methods:

***main()*: void** - establish connection and generate Dashboard screen

***connect()*: Connection** - test connection to database and pass it to *main()*

Main class has not been implemented in model, because it wasn't thought of during modeling. Although this class in very small, it is vital for the functionality of system, since it establishes connection to database and invokes Dashboard. Creating one connection which is then passed between screens may seem to be inefficient, but in actuality, it is much more feasible than establishing new connection every time the database needs to be queried and subsequently closing the connection.

Dashboard

Dashboard class displays welcome screen with trending auctions and options to sign in or sign up. When sign up is selected, a new window appears where user can insert personal information and select whether he wants to be a Bidder or an Organizer. User also has to choose whether he inserted ID number (*OP*) or ID of his company (*IČO*). All fields need to be filled in or a warning is produced. After confirming registration, all fields are validated and new user is created in database which is subsequently logged in. Note that only Bidder and Organizer can sign up this way, Operator has to be added through database.

When sign in is selected, new dialog window is produced for username and password. After confirming login, information is passed to database query which determines whether the user exists and the credentials are valid. If username and password match, the user is logged in, which means that the information about said user are queried from database and the data is saved to *loggedIn* variable, otherwise dialog produces a warning that the username or password don't match. After login, new buttons are created according to user class (Bidder, Organizer, Operator), as well as logout button: Organizer can create new auction, so on top left is created button for such purpose, and Bidder can report auction, so on the same place is created button to issue a report, however reporting function is not implemented in this demonstration program.

Under the top bar is a list of auctions ordered by name. After clicking on auction name, a new window with auction information is opened. Functionality of this window is further described in section about Auction class.

Variables:

frame: JFrame - window

canvas: DashboardCanvas - JPanel for drawing graphical objects

connection: Connection

loggedIn: User

components: ArrayList<JComponent> - container for buttons created according to user class

Methods:

***Dashboard*(*conn*: Connection, *user*: User)** - constructor for Dashboard, *conn* is saved to *connection* and *user* is saved to *loggedIn*. If *user* is *null*, it means that no one is logged in and the buttons for sign in and sign up are produced

***generateDashboard()*: void** - generate Dashboard screen

***parseUser(rs: ResultSet)*: void** - parse data about user from database and determine user class. Data is saved in *loggedIn* variable, which insinuates that a user has logged in.

***createComponents*(): void** - create buttons according to user class and save them to *components* variable, from which they are subsequently added to window

***parseHTML*(*color*: String, *size*: String, *text*: String): String** - prepare string in HTML format to easily change color, size, font and contents of text in labels and buttons.

class *DashboardCanvas* - JPanel for drawing graphical objects on screen

Dashboard is not implemented as a class in model, only as an interface. However, it is much more viable to use it as class, so it can contain variables and methods. Model doesn't account for the use of *loggedIn* and *connection* variables, which are imperative for correct functionality of program. As it was previously mentioned in Main class, *connection* could be established every time the database needs to be queried and then closed, but it's inefficient and the name of the database needs to be only supplied once, to *connect*() method in Main class. Otherwise the name would have to be hard-coded, which would make modifications to program difficult, or pass the name of database every time it is needed to access it, which is very inefficient.

loggedIn is important to keep track of the User that is currently logged in, which wasn't thought of in model, so it had to be added in the program as well as the *connection* variable.

Auction

Auction class contains all important information about every e-auction in system. When an auction is selected from Dashboard, its data is queried from database and new object is created.

Auction can produce two kinds of windows - information and creation. Information window displays all information about Auction, name, description, charity it supports and start and end dates of auction. Under basic information is list of biddings with starting bid (if no one made a bid yet) or highest bid (if someone already has made a new bid). After selecting a Bidding, a new window is produced that is further explained in section about Bidding class.

Creation window contains fields for all basic information, as well as drop-down menu of available charities. Again, all fields have to be filled in, or a warning is produced. After confirming creation of Auction, the information is validated and saved to database, and new window for creating Biddings is produced. Again, this window is further explained in next section.

Variables:

frame: JFrame - window
canvas: AuctionCanvas - JPanel for drawing graphical objects
connection: Connection
loggedIn: User

id: int
organizerId: int
charityId: int
name: String
description: String
auctionStart: Datetime
auctionEnd: Datetime

Methods:

***Auction*(*user*: User, *conn*: Connection, *id*: int, *organizerId*: int, *charityId*: int, *name*: String, *description*: String, *auctionStart*: Datetime, *auctionEnd*: Datetime)** - constructor for existing *Auction*, all arguments are saved to respective variables

***Auction*(*user*: User, *conn*: Connection)** - constructor for new *Auction*, which also calls *generateAuctionCreation()*

***generateAuctionInfo()*: void** - generate screen with *Auction* information

***generateAuctionCreation()*: void** - generate *Auction* creation screen

***save*(): void** - create new statement and insert information from *Auction* creation window to database

***parseHTML*(*color*: String, *size*: String, *text*: String): String** - prepare string in HTML format to easily change color, size, font and contents of text in labels and buttons.

***get*()** functions

class *AuctionCanvas* - JPanel for drawing graphical objects on screen

Auction takes method to create new Auction from Organizer class in model. Even though Organizer invokes said method, the window and inserted information is mostly related to Auction, save for information about Charity, Biddings and Organizer, so it's more reasonable to move the method to this class.

Bidding

Bidding contains information about one separate bidding in auction. Like Auction, Bidding also can produce two types of window - information and creation. Information window again shows information about bidding, name of Auction the Bidding appears in, starting bid and if exists, also highest bid and username of the highest bidder. When Bidder is logged in, a button to make a new bid is created as well. When user selects it, a new dialog window is produced with the current highest bid and field to insert new bid. The bid has to be higher than the previous one, or a warning is produced. When the new bid is valid, the bid is saved to database along with the id of the new highest bidder and the bidding is reloaded.

Bidding creation window contains fields for all required information - name, description, starting bid, start and end of bidding. Then Organizer has an option to save and create new bidding, or save and finish creating biddings. All biddings are saved to database and after reloading Dashboard, they are also visible in Auction information screen.

Variables:

frame: JFrame - window
canvas: BiddingCanvas - JPanel for drawing graphical objects
connection: Connection
loggedIn: User

id: int
auctionId: int
name: String
description: String
startingBid: float
highestBid: float
highestBidderId: int
biddingStart: Datetime
biddingEnd: Datetime

Methods:

***Bidding(conn*: Connection, *user*: User, *id*: int, *auctionId*: int, *name*: String, *description*: String, *startingBig*: float, *highestBid*: float, *highestBidderId*: int, *biddingStart*: Datetime, *biddingEnd*: Datetime)** - constructor for existing *Bidding*, all arguments are saved to respective variables

***Bidding*(*conn*: Connection, *auctionId*: int, *user*: User)** - constructor for new *Bidding*, which also calls *generateBiddingCreation(). auctionId* is passed to save the right id to database

***generateBiddingInfo()*: void** - generate screen with *Bidding* information

***generateBiddingCreation()*: void** - generate *Bidding* creation screen

***save*(): void** - create new statement and insert information from *Bidding* creation window to database

***parseHTML*(*color*: String, *size*: String, *text*: String): String** - prepare string in HTML format to easily change color, size, font and contents of text in labels and buttons.

***get*()** functions

class *BiddingCanvas* - JPanel for drawing graphical objects on screen

class BiddingCreationCanvas - JPanel extending *BiddingCanvas* for drawing graphical objects on screen

Just as with Auction, method to create new Bidding has been moved from Organizer class, which offers easier access to variables. Other variable that has not been modeled is an object of JFrame class, which encapsulates all graphical objects, fields and buttons. This class is special for Java and other programming languages may have different representations of graphical interface, so the *frame* variable is left out on purpose to not confuse programmers in case they would implement the software in other programming language than Java.

User

User contains temporary information about logged in user, or User who has just registered. User also has three extending classes: Bidder, Organizer and Operator. While Bidder and Organizer can be registered through program, Operator needs to be added through database.

User can specify whether he wants to be Bidder or Organizer during registration. Registration is performed through *createUser*() method which generates new window. User inserts all required information, specifies whether he is registering a person or company (*OP* insinuates person, *ICO* company) and after validation, a new User is created and added to database. Said User is also subsequently logged in and can create (in case user registered as Organizer) or participate (registered as Bidder) in Auctions.

Class also contains prepared methods for modifying User information, however this functionality is not fully implemented in the program. These methods are *setPersonal*() for modifying first and last name, *setUsername*() for changing username and *setPassword*() for changing password. Each of these functions evaluates input and saves it to database into respective column.

Variables:

frame: JFrame - window
canvas: UserCanvas - JPanel for drawing graphical objects
connection: Connection

id: int
firstName: String
lastName: String
username: String
password: String - saved only if new User is created
addressLine: String - String constructed from *street*, *city*, *state* and *zip*
OP: String - personal ID
ICO: String - company ID

paymentMethods: PaymentMethod - only in *Bidder* and *Organizer* class, container for registered payment methods
bookmarks: Auction - only in *Bidder*, container for bookmarked Auctions

Methods:

***User(conn*: Connection, *id*: int, *firstName*: String, *lastName*: String, *username*: String, *password*: String, *street*: String, *city*: String, *state*: String, *zip*: String, *op_ico_number*: String, *op*: boolean)** - constructor for existing *User*, all arguments are saved to respective variables, *addressLine* is constructed from *street*, *city*, *state* and *zip* and *op* determines whether *op_ico_number* is saved into *OP* or *ICO* (if true - OP, if false - ICO)

***User(conn*: Connection)** - constructor for new *User*, which also calls *createUser()*

***createUser()*: void** - generate *User* creation screen

***setPersonal(firstName*: String, *lastName*: String*)*: void** - save new first and last name to database

***setUsername(username*: String*)*: void** - save new username to database

***setPassword(password*: String*)*: void** - save new password to database

***save*(): void** - create new statement and insert information from *User* creation window to database

***parseHTML*(*color*: String, *size*: String, *text*: String): String** - prepare string in HTML format to easily change color, size, font and contents of text in labels and buttons.

***get*()** functions

class *UserCanvas* - JPanel for drawing graphical objects on screen

Datetime

Datetimes in this system are very important since every Auction and Bidding needs to contain data about start and end. That can be easily represented by *Date* class from *java.util* package, but it makes working with datetimes easier if they are implemented just as they are required to work. Also, since SQLite can't work with datetimes, they are represented by strings. That means the date has to be converted to string in a format that can be subsequently read and parsed. Use of PostgreSQL database wouldn't solve the problem, because the datetime would be returned as string from the database and would still need to be parsed.

Datetime implements multiple overloaded constructors for different uses. It can parse date from integers for every part of datetime, from string and from *Date* class, which is used when creating *createdAt* and *updatedAt* fields in database, since Date can easily determine current datetime.

Possible point of extension is working with timezones for which is also prepared constructor, so the dates can be converted to the timezone the user is in. Another thing to note is the use of int for *second* variable, instead of float or double, as it is common with Date classes. The reason is, that although the start and end of Auction or Bidding has to be precise, it doesn't need to work with decimal numbers and integers work just fine.

Variables:

year: int
month: int
day: int
hour: int
minute: int
second: int
timezone: String

Methods:

***Datetime(year*: int, *month*: int, *day*: int, *hour*: int, *minute*: int, *second*: int, *timezone*: String)** - constructor for *Datetime* with timezone

***Datetime(year*: int, *month*: int, *day*: int, *hour*: int, *minute*: int, *second*: int)** - constructor for *Datetime* without timezone

***Datetime(datetime*: String)** - constructor for *Datetime* from String. This constructor is used to parse data from database and fields in creation screens

***Datetime(datetime*: Date)** - constructor for *Datetime* from java.util.Date. This constructor is used when saving *createdAt* and *updatedAt* fields to database

***getDateString()*: String** - construct String from *Datetime* in format “yyyy-mm-dd hh:mm:ss”

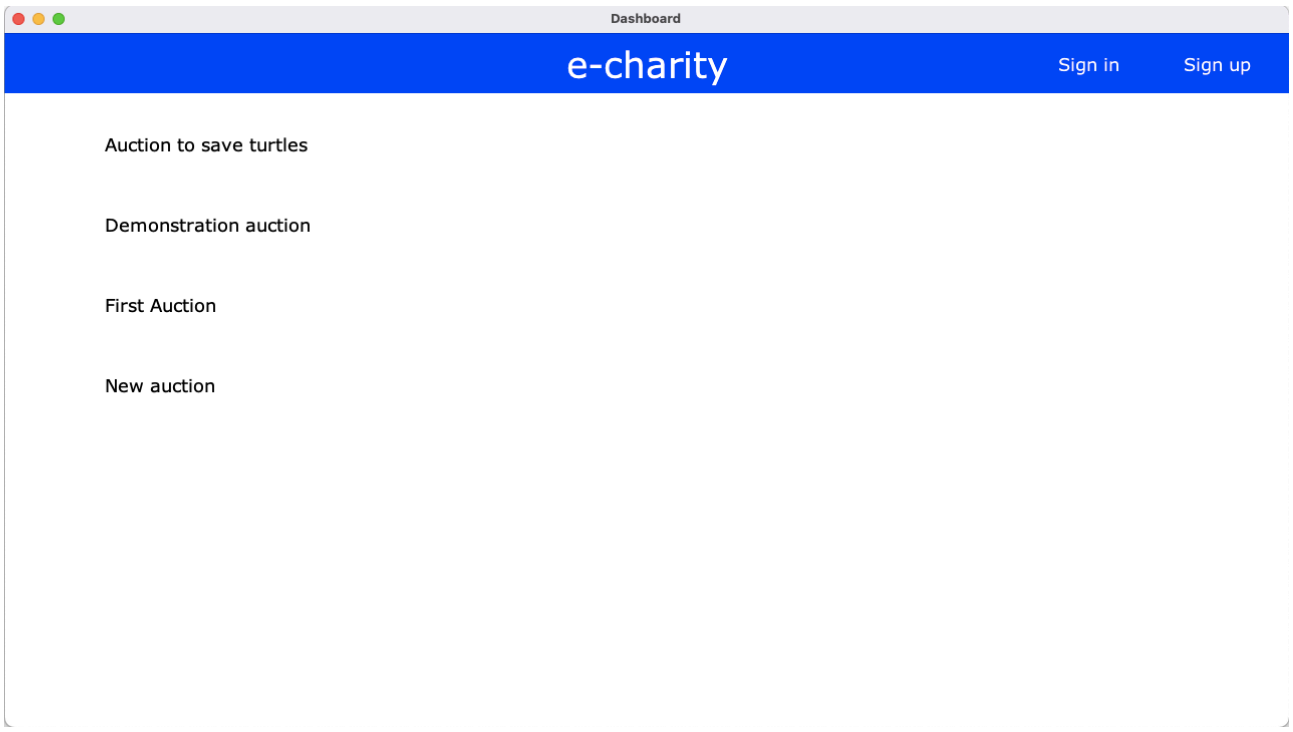
***isLower(d*: Datetime*)*: boolean** - compare two *Datetimes* and return true if *this Datetime* is lower than the *d* argument

***get*()** functions

System also has prepared multiple other classes like PaymentMethod and Report, which are not fully implemented. However, the basic functionality of system is there - creating Auction and Bidding, registering new User, viewing information about Auctions and Biddings and making a new bid. All of these functions are also implemented with the use of database, so all changes are persistent.

Screenshots:

As was mentioned before, the first screen that appears is Dashboard. In the top bar are two buttons to sign in, or sign up and under the bar is a list of auctions:



Auction to save turtles

Demonstration auction

First Auction

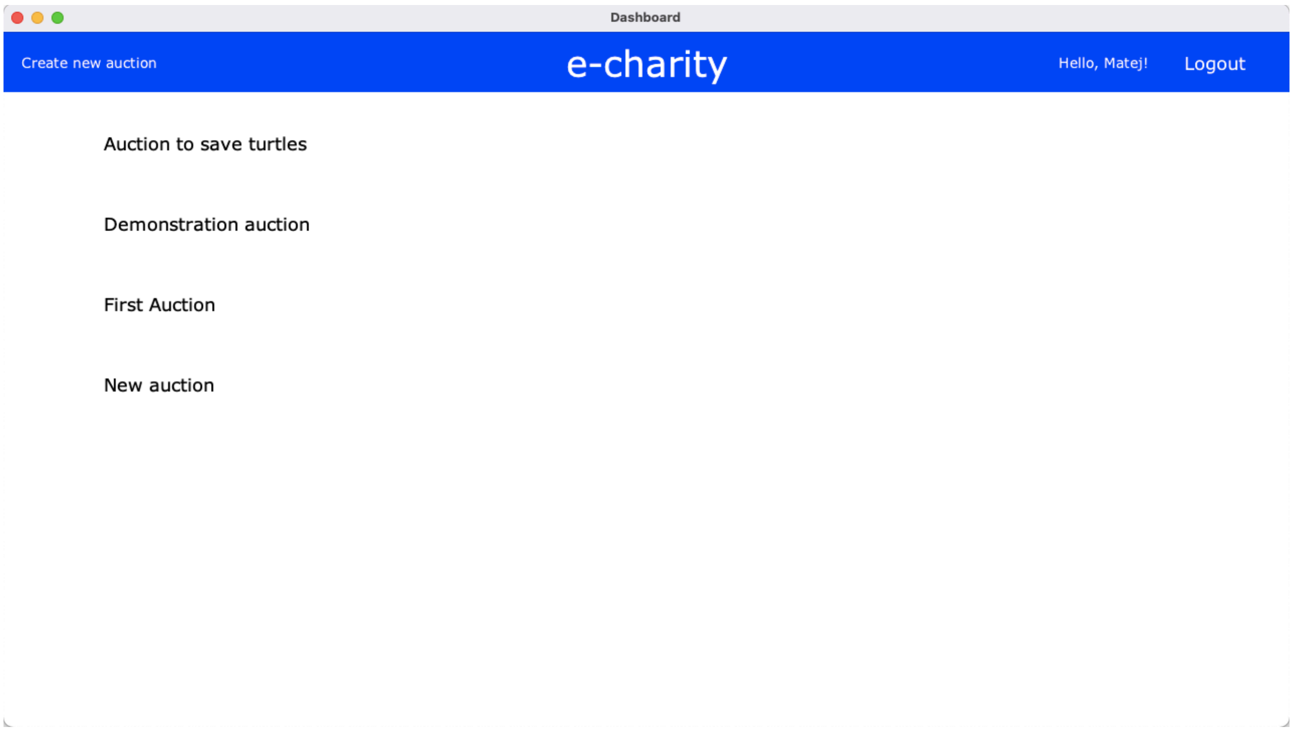
New auction

Login credentials:

Operator: **admin** and **ADMINPASS**,

Organizer: **mhrnciar** and **password**,

Bidder: **companyaccount** and **company123**


Dashboard

Create new auction

e-charity

Hello, Matej!

Logout

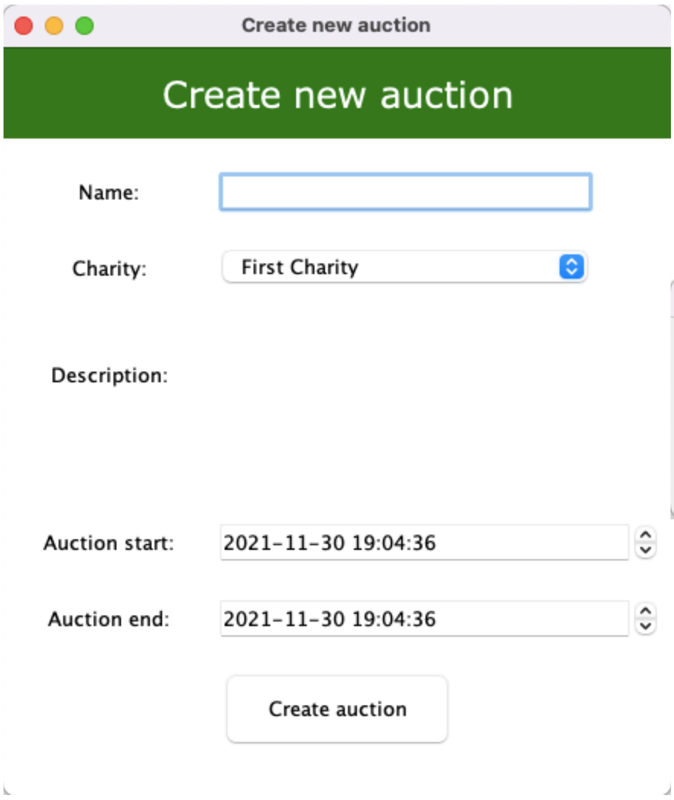
Auction to save turtles

Demonstration auction

First Auction

New auction

Let’s say we logged in as Organizer. On the left appeared new button to create a new Auction. After pressing said button, a new window is produced:


Create new auction

Create new auction

Name:

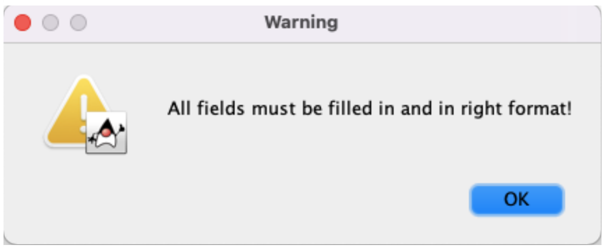
Charity: First Charity

Description:

Auction start: 2021-11-30 19:04:36

Auction end: 2021-11-30 19:04:36

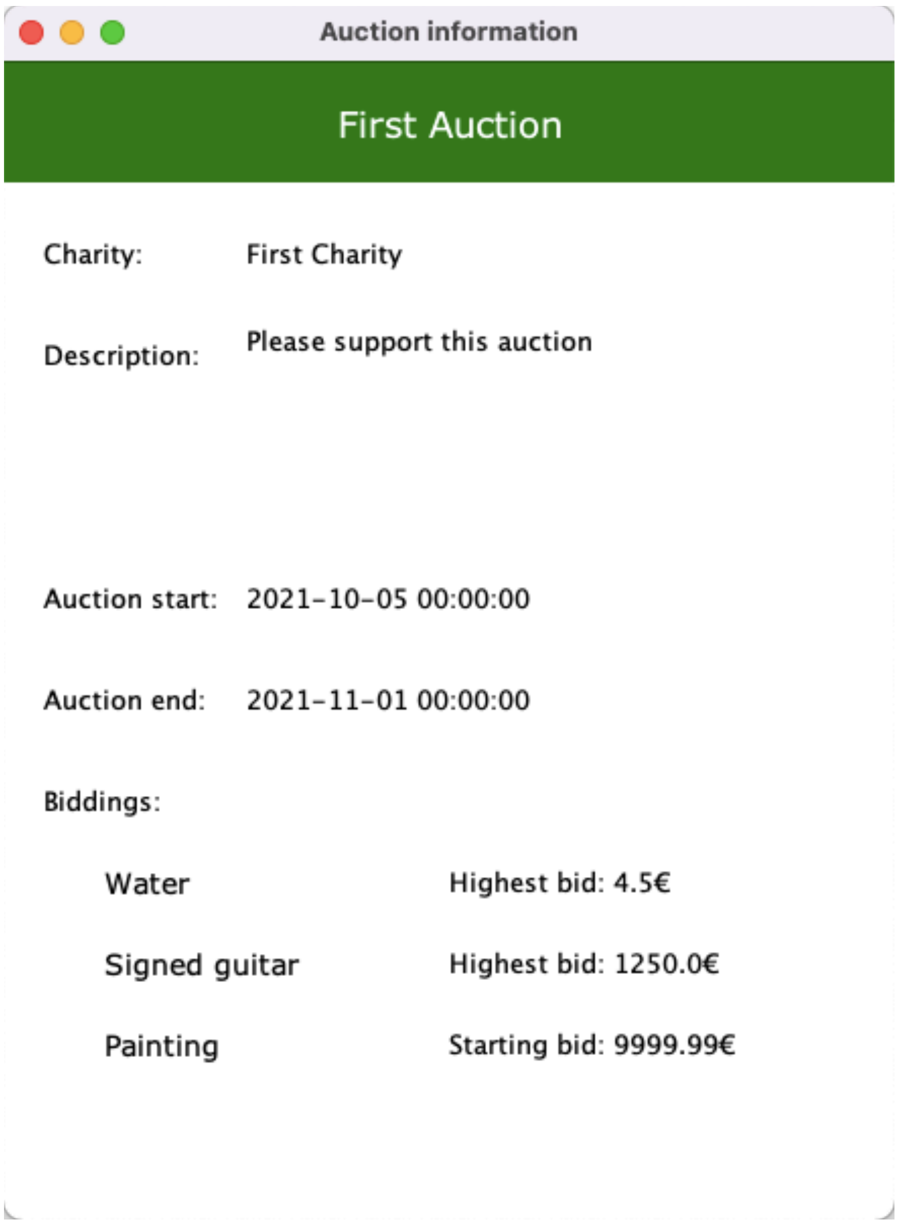
Create auction


Warning

All fields must be filled in and in right format!

OK

All fields must be filled in, or a warning displayed above will appear. This warning also appears during login, registration and creating a new bidding. Now, let's log in as a Bidder. After clicking on Auction from list in Dashboard, a new window opens with information about Auction and a list of Biddings:



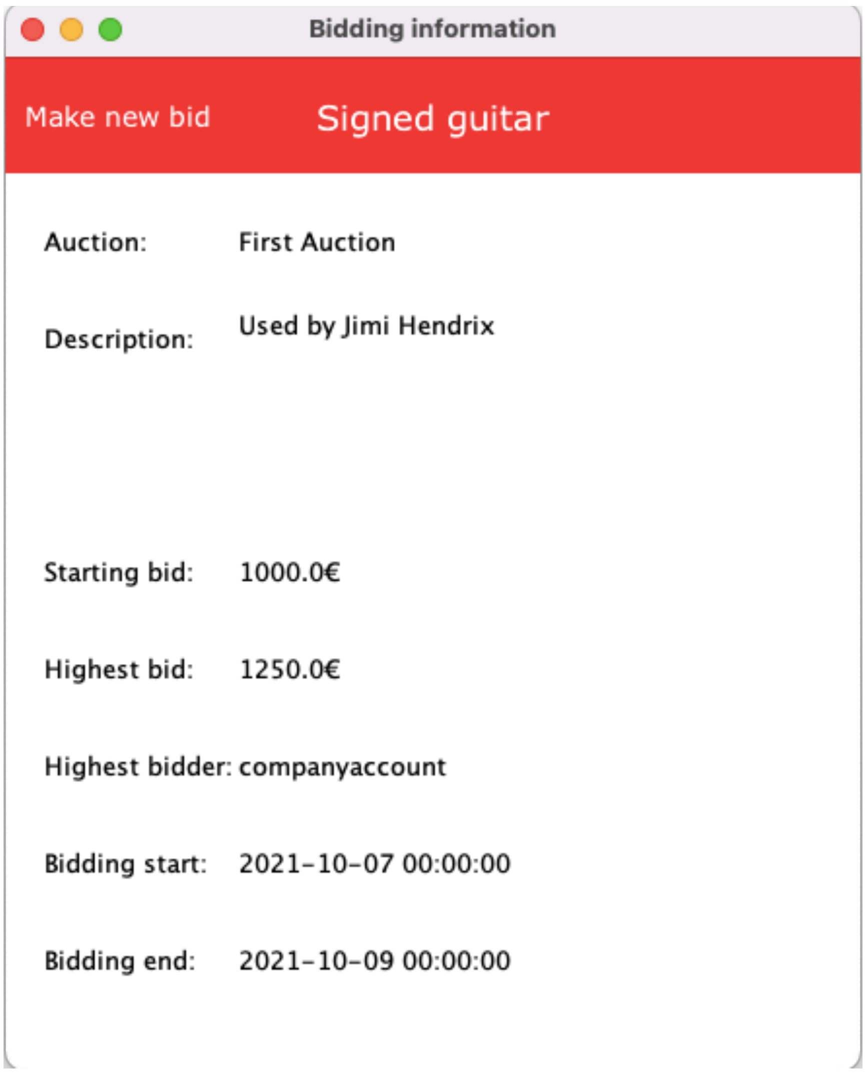
First Auction

Charity:	First Charity
Description:	Please support this auction
Auction start:	2021-10-05 00:00:00
Auction end:	2021-11-01 00:00:00

Biddings:

Water	Highest bid: 4.5€
Signed guitar	Highest bid: 1250.0€
Painting	Starting bid: 9999.99€

We can see that the first two biddings have already been participated in, since there isn't starting bid, but highest bid. When we click on one of the biddings, a window with bidding information opens. If the user is logged in as a Bidder, a button to make a new bid appears on the left of the top bar. After clicking on the button, a dialog window opens where Bidder can insert new highest bid and after confirming the bid, information on the screen is updated to show the new bid and the username of the highest bidder:



Signed guitar

Auction:	First Auction
Description:	Used by Jimi Hendrix
Starting bid:	1000.0€
Highest bid:	1250.0€
Highest bidder:	companyaccount
Bidding start:	2021-10-07 00:00:00
Bidding end:	2021-10-09 00:00:00