

Bidster
A Blind Dutch Auction Ticket Marketplace
Ciaran Doherty
20475234
Final Year Project – 2024
B.Sc. Single Computer Science and Software Engineering



**Maynooth
University**
National University
of Ireland Maynooth

Department of Computer Science
Maynooth University
Maynooth, Co. Kildare
Ireland

A thesis submitted in partial fulfilment of the requirements for the B.Sc.
Single Computer Science and Software
Engineering.

Supervisor: Barak A. Pearlmutter

USER MANUAL

Table of Contents

<i>Introduction</i>	2
<i>Blind Dutch Auction. How does it Work?</i>	2
Setting Up And Running Bidster	3
<i>Getting Started</i>	4
Creating an Account	4
Browsing Events	5
Placing Bids	6
User Profile	8
Logging Out	8
Continue As Guest.....	8
Admin Privileges.....	8
<i>Software Design Documentation</i>	9
Frontend Design.....	9
Key Frontend Components are:.....	9
React and Material-UI.....	10
Notifications and Alerts	11
Navigation and Routing	11
Backend Design	11
Technologies Used	11
Database Structure.....	11
REST API Endpoints.....	12
<i>Key Backend Functions</i>	13
Process Bids()	13
Step-by-Step Functionality	14
Error Handling and Transactions.....	14
Scheduled Processing	14
GenerateTickets().....	14
Step-by-Step Functionality	14
CheckAuctionStatus().....	15
<i>FAQs</i>	15

Introduction

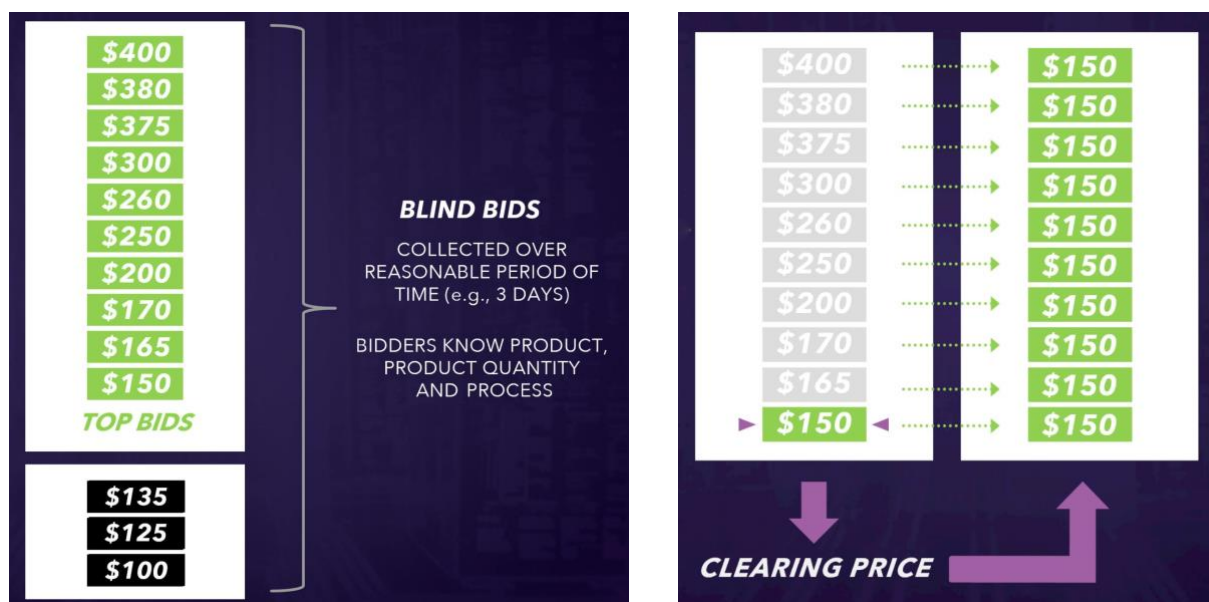
Welcome to Bidster, the modern-day solution to revolutionizing the ticket marketplace. Bidster offers a unique approach to the marketplace through Blind Dutch Auction, which aims to innovate the current domain by designing a website which promises greater fairness, efficiency and transparency in ticket sales. Furthermore adding an aspect of excitement to the whole process. The Blind Dutch Auction model achieves this by allowing buyers to place bids on tickets, bidding the maximum they're willing to pay per ticket, with the auction being 'Blind' meaning the bids from buyers are not disclosed to other participants until the auction is concluded and the winners are determined. The conclusion of this model is that all bids are gathered and sorted from highest to lowest. The top N number of bids (N being the total amount of tickets available) wins their tickets, but all pay the exact same price, the Clearing Price, which is the lowest bid of the winning Bids.

Blind Dutch Auction. How does it Work?

Here's how the Process Works:

1. **Live Auctions:** Each event has its own auction and may run for a period of days. Buyers have the freedom to bid the max they are willing to pay for a ticket within this time period.
2. **Placing a Bid:** When a Buyer places their bid, their payment method is temporarily placed on hold for the amount they have bid. They will only be charged in the event that they win their Bid.
3. **Auction Closure:** When the auction finishes, all the bids are processed and the highest bids are matched with the number tickets of available. The lowest of these Bids becomes the Final Price for all Winners, otherwise called 'The Clearing Price'.
4. **Notification:** Win or Lose, the Buyers will be informed of the outcome of their bid, and only the winners will be charged.

For Example: If a buyer Bids €50/ticket for 2 Concert Tickets and the Clearing Price ends up at €30/ticket, the buyer only pays the Clearing Price of €30/ticket and will be charged €60 in total. But if the buyer bids €25/ticket, the buyer loses and won't be charged at all.



Setting Up And Running Bidster

This section is the setup guide for running Bidster locally on your machine. The following steps will guide you through the process of setting up the environment needed to run Bidster. The application is built on a React frontend and MySQL backend, requiring the use of XAMP to manage the Database locally.

Step 1: Installing XAMPP

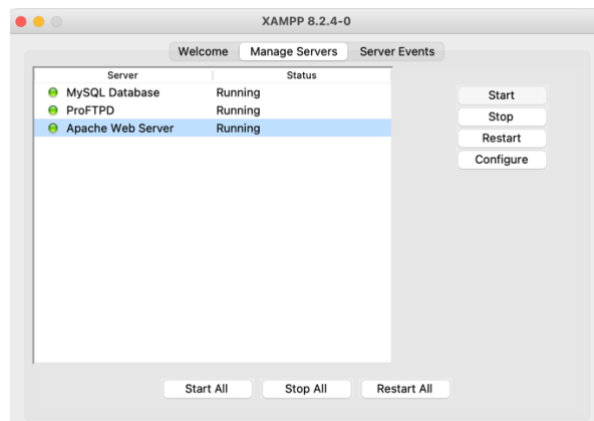
XAMP is a free and open-sourced cross-platform web server that will allow you to run the MySQL database locally on your machine.

1. **Download XAMPP:** Go to the official XAMPP website and download the version most compatible with your operating system.
2. **Install XAMP:** Open the downloaded file and follow the installation instructions, making sure to the MySQL and phpMyAdmin components.

Step 2: Setting Up Database

Once you have installed XAMPP you will need to create the database for Bidster using the Database Dump I have provided with the project folder.

1. **Start XAMPP:** Launch the XAMPP Control Panel and start the MySQL Database and the Apache Web Server.



2. **Go to phpMyAdmin:** Open your Browser and go to 'http://localhost/phpMyAdmin'.
3. **Create Database:** Click on the 'Import' Tab at the top. Under the "Files to Import" section, click "Choose File" and select the Database Dump file provided with the project. Click "Go" at the bottom of the screen to create the database.

Step 3: Installing Node.js

1. **Download Node.js:** Go to the official Node.js website and download the version compatible with your operating system.
2. **Install Node.js:** Open the Downloaded File and follow the installation instructions.

Step 4: Setting Up the Project

1. **Download Project or Clone Project:** Download or clone the project from Git Repository.
2. **Install Dependencies:** Open the Terminal in the root directory of the project and run the following command to install the required dependencies: `"npm install"`

Step 5: Running the Application

After successfully completing the previous steps, you are now ready to run Bidster. Bidster is run off two folders. Frontend and Backend.

1. **Backend Directory:** From the root directory navigate to the backend folder “*cd backend*”.
2. **Starting the Backend Server:** Run “*npm start*” or “*npm run dev*” (for Development purposes). This will start the Server.
3. **Frontend Directory:** Also from the root directory navigate to the frontend folder “*cd frontend*”.
4. **Start React Frontend:** Run “*npm start*”. This will start the React Application and open the Browser automatically.

You should now see Bidster running on your Browser. You can now explore the functionalities of Bidster.

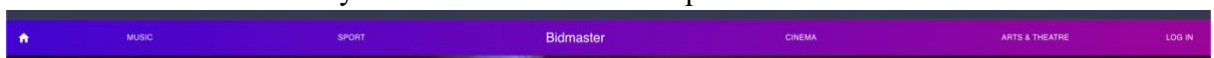
Getting Started

This section will guide new Users through the initial steps of creating an account and how to use the overall features of Bidster. Features such as browsing events, placing bids and updating their details, ensure that users have the knowledge to navigate through the website easily and effortlessly.

Upon visiting Bister, the user is greeted with a welcome page introducing the Blind Dutch Auction and its rules. Here’s how the user may navigate the site and take the first steps towards participating in Auctions.

Creating an Account

1. **Navigating to the Sign In/Sign Up Pages:** Users can navigate to the log-in through the ‘Login’ button in the header. New users, who intend on placing bids should click on the “Don’t have an Account? Sign Up” Link to be directed to the register page. Guests will have the ability to browse events but not place Bids.



2. **Registration Form:** The Registration Form requires users to input their First Name, Last Name, Email Address and Password. Upon a successful registration a notification should appear confirming the registration, the user will then be navigated to the Sign In Page to Log in.



Register

First Name* Last Name*

Email Address*

Password*

[Sign Up](#)

[Already have an account? Sign In](#)

3. **Sign-In Form:** The Sign-in form requires users to input their Username/Email and Password. Once filled out the user may either click on the sign-in button or hit Enter. If user details are correct they will be navigated back to the homepage.



Sign In

Email Address*

Password*

[Sign In](#)

[CONTINUE AS GUEST](#)

[Demo User](#) [Don't have an account? Sign Up](#)

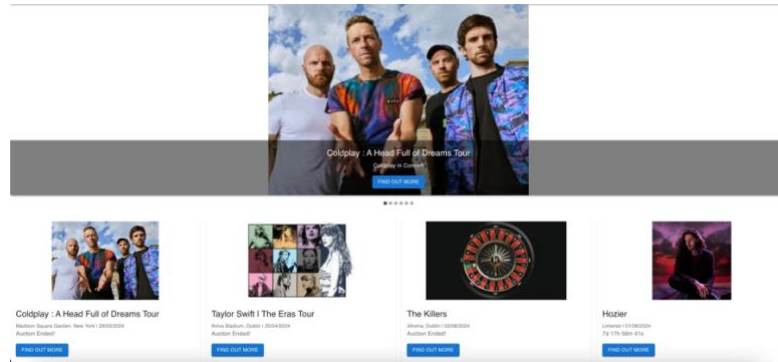
Browsing Events

Users can browse through various events listed on the Site. These events are sorted based on their Categories, such as Cinema, Music, Sport and Theatre.

1. **Selecting Event Categories:** The different categories are listed in the Headers Menu Bar. Click on a Category and the User will be navigated to their pages displaying all their Events.




2. **Browsing Events:** Featured Events are displayed at the top of the page in a carousel, with all the events displayed underneath neatly styled as cards, showing the events info and a countdown clock dictating how long is left to go in the Auction. To see more details about the event or to place a bid on the event, the user can click on the 'Go to Event' button located at the bottom of each event card.



3. **Expired Events:** Expired events will still be displayed for a period of time. To indicate which events are expired, in place of the countdown clock it will read 'Auction Ended'. The User can see the auction's end results by clicking on the "Go to Event" button located at the bottom of the event card. If a user is still unfamiliar with how auctions work, this is a good way of familiarizing themselves before placing their first bid.

Coldplay : A Head Full of Dreams Tour



Madison Square Garden, New York | 29/03/2024

Place your bid

Max Bid per ticket: €

Quantity:

PLACE BID

Time Remaining
Auction Ended!

Auction Results:
Final Price: €268


- 1. Matty@gmail.com: €464
- 2. claran.doherty.2021@mumail.ie: €422
- 3. JaneDoe@gmail.com: €384
- 4. Dego@gmail: €364
- 5. Jack@gmail.com: €268
- 6. JohnDoe@gmail.com: €183
- 7. Aidan@gmail.com: €157
- 8. admin@admin.com: €55

Placing Bids

To place a bid:

1. **Select an Event:** Users choose an event they are interested in and that is still in progress.
2. **Review Events Details:** User reviews event details and auction T&Cs.
3. **Enter a Bid:** User enters the amount they're willing to pay per ticket and the quantity of tickets they're looking to bid on in the bidding form on the top right-hand side of the page.
4. **Place Your Bid:** User clicks on the 'Place Bid' button and will be presented with a notification stating a successful bid or unsuccessful bid.

Hozier



Limerick | 01/08/2024

Place your bid

Max Bid per ticket

€

Quantity

1

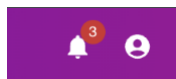
PLACE BID

Time Remaining

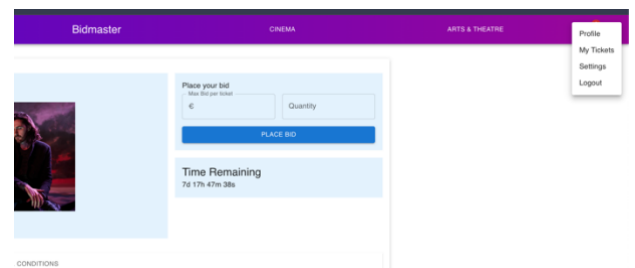
7d 17h 51m 10s

A Bid may be Unsuccessful if:

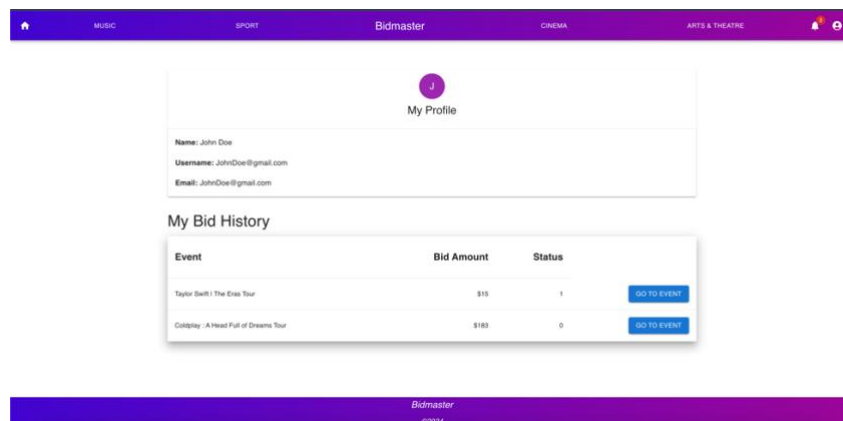
- The user is not logged in
 - The user didn't fill out all the fields
 - The user didn't bid a high enough amount (Organisers have the ability to place a Minimum Bid on their auctions).
 - The User is not connected to the Internet/Server
 - The Auction has already concluded.
5. **Notifications:** User will receive a notification stating their bid results. These notifications can be seen and read in the bell icon on the header.



6. **My Tickets:** If successful the user's tickets can be found in the 'My Tickets Page', which is accessible in the profile icons dropdown menu in the header.



7. **Bid History:** User may also view their bid history in the 'My Profile Page'. They can view all their past and current bids and their outcome.



User Profile

Users can access their account details in the My Profile Page, accessible in the header from the profile icon dropdown menu.

Logging Out

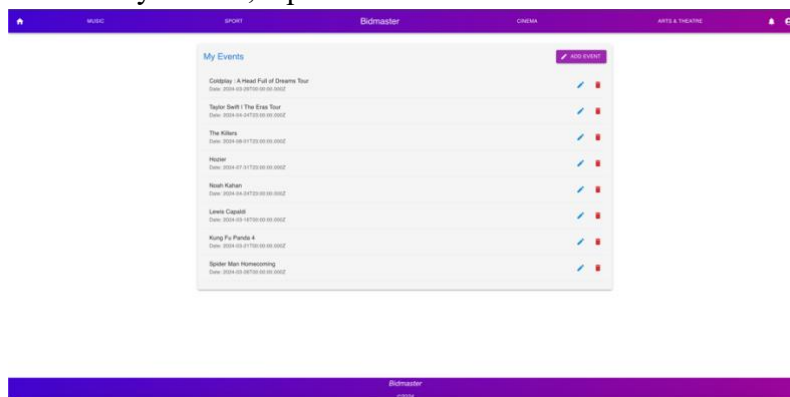
Users can log out by selecting the 'Logout' button in the profile icon dropdown menu in the header. The user will then be navigated back to the sign-in page.

Continue As Guest

By continuing as Guest on the sign-in page, the user has the ability to view all the events and auctions on offer, but will not have the ability to place bids on any events.

Admin Privileges

Users and organisers can have Admin Privileges which give them access to the admin pages, which give them the ability to Add, Update and Delete Events.



1. **Becoming Admin:** At this time to become Admin it must be manually updated in the database at the request of the site admin.
2. **Navigating Admin Pages:** Admin pages can be accessed through the profile icon drop-down menu in the Header.
3. **Admin Page:** The Admin page contains details of all events and auctions currently on the site, with the options of adding new events, updating current events and deleting Events.
4. **Add New Events:** To Add a new Event click on the “Add Event” button on the top right-hand side, this will navigate you to a form for inputting event details and auction rules.
5. **Edit Events:** To edit current events click the edit icon button beside the event you wish to update.
6. **Delete Events:** To delete an event click the Trash Can Button beside the event you wish to delete.

Software Design Documentation

This section provides the technical aspects of Bidster, and a detailed overview of the architecture, including the Frontend and Backends components.

Frontend Design

The frontend of the application is built using React, offering a dynamic and responsive user experience. Material UI is also utilized to provide a modern look to the user interface. For communication between the frontend and backend, Axios is used for handling RESTful API calls such as registering users, creating events, and placing user bids etc.

Key Frontend Components are:

- **App.js:** This manages the page navigation and access using “react-router-dom” for routing. The routing is separated into two main categories based on access level: Public and Protected. Public Routes are accessible to all users, regardless of their authentication status. Protected Routes require user authentication such as logging in or admin status.

```
function App() {
  return (
    <BrowserRouter>
      <ReactNotifications/>
      <Routes>
        {/}SIGN IN/ REGISTER/ USERS {/}
        <Route>
          <Route path="/users" element={}<Users/> } />
          <Route path="/login" element={}<LoginForm/> } />
          <Route path="/register" element={}<Signup/> } />
          <Route path="/authorization-denied" element={}<AuthDenied/> } />
        </Route>
        {/}PUBLIC PAGES {/}
        <Route element={}<PageLayout/> } />
        <Route path="/" element={}<Home /> } />
        <Route path="/music" element={}<Musicpage/> } />
        <Route path="/cinema" element={}<Cinemapage/> } />
        <Route path="/sport" element={}<Sports/> } />
        <Route path="/arts&theatre" element={}<ComingSoon/> } />
        <Route path="/eventpage/:event_id" element={}<Eventpage/> } />
      </Route>
      {/}USERS ONLY PAGES {/}
      <Route element={}<ProtectedRoutes/> } />
      <Route element={}<PageLayout/> } />
      <Route path="/myprofile" element={}<MyProfile/> } />
      <Route path="/settings" element={}<Settings/> } />
      <Route path="/addevent" element={}<AddEvent/> } />
      <Route path="/events/edit/:event_id" element={}<EditEvent/> } />
      <Route path="/mytickets" element={}<MyTickets/> } />
    </Route>
    {/}ADMIN ONLY PAGES {/}
    <Route element={}<AdminRoute/> } />
    <Route element={}<PageLayout/> } />
    <Route path="/admin" element={}<Admin/> } />
  </Route>
</Route>
</Route>
</Routes>
</BrowserRouter>
```

- **PageLayout.js:** This component is used to provide consistent structure and layout throughout the application. It wraps around all the pages and includes the Header and Footer components ensuring that the navigational links are accessible throughout the application when necessary.
- **Header.js:** The Header component holds the functionality for navigating throughout the application, receiving Notifications and Logging in and Out. The header is created with Material-UI components such as “App Bar”, “Tool Bar”, “Menu”, “MenuItem” and Icons such as the “HomeIcon”, “AccountCircle”, and “NotificationsBadge”. It uses Reacts States and Hooks “useState()” and “useEffect()” to fetch and mark Notifications as “read” using Axios. Functions such as “handleClose()” for the functionality of the dropdown menus and Logging in and Out are also in this

component. The component is also conditional for situations where the User is and isn't logged in.

- **Home.js:** The Home Page is the first page that Users are greeted with it provides information about the site.
- **Concertpage.js:** This acts as a reusable component for all the event categories, displaying featured events in carousels and displaying all the events for browsing. It uses *axios.get()* for getting Events from the Database and displays them using Material-UI's Card components. The carousel is imported from the react library "react-material-ui-carousel".
- **Eventpage.js:** This component holds some of the core functionalities of the application. It serves as the interface for viewing individual Events Details, Auction Details, Auction Results and the ability to place Bids. It uses React State and Hooks, "useState()" and "useEffect()" to manage data, bidding logic and real-time updates. The use of Material-UI components such as 'Container', 'Box', 'Grid' and 'Typography' creates a well-structured and visually appealing design and layout. The "Countdown" component is integrated to display the remaining time left in the auction, improving user engagement with time-sensitive actions such as placing bids.
- **Login.js & AuthContext.js:** The Login component manages user authentication, including signing in and continuing as a guest functionality. It works with the backend to authenticate user's Axios for requests to the Database. AuthContext provides a context for the user authentication state, managing which users are logged in and updating the browser's Local Storage and Session Storage.

```
//context for managing authentication
const AuthContext = createContext();

//hook for useContext
export const useAuth = () => useContext(AuthContext);

//Admin route to only allow users with admin role
export const AdminRoute = () => {
  const {currentUser} = useAuth();
  const isAdmin = currentUser && currentUser.Role === 'admin';
  return isAdmin ? <Outlet/> : <Navigate to='authorization-denied'/>
}

//Protected Routes for pages that you need to be logged in for
export const ProtectedRoutes = () => {
  const isLoggedIn = !!localStorage.getItem('currentUser');
  return isLoggedIn ? <Outlet /> : <Navigate to='/login' />;
};

//Provides Authentication and manages user state
export const AuthProvider = ({ children }) => {
  //sets current user from local storage so it maintains its session on reloads
  const [currentUser, setCurrentUser] = useState(() => {
    const userJson = localStorage.getItem('currentUser');
    return userJson ? JSON.parse(userJson) : null;
  });
  //useEffect to set currentUser in localStorage
  useEffect(() => {
    // Sync currentUser state with localStorage/sessionStorage
    if (currentUser) {
      localStorage.setItem('currentUser', JSON.stringify(currentUser));
    } else {
      localStorage.removeItem('currentUser');
    }
  }, [currentUser]);

  //function to login users and updates currentUser
  const login = (userDetails) => {
    setCurrentUser(userDetails);
  };
};
```

React and Material-UI

The Application utilizes React and Material-UI extensively throughout. The use of React Hooks ("useState(),"useEffect()") for its state management and backend operations. Material

UI is integrated with React to customise and create responsive components. This can be seen throughout the application in all the Pages, Forms, Buttons and Navigation Tabs. This ensures an appealing aesthetic throughout the application.

Notifications and Alerts

Real-time Notifications and Feedback are delivered using 'react-notifications-component'. Providing Users with feedback from their actions such as a Successful Bid Placement or Winning in an Auction. These notifications enhance the user's overall user experience.

Navigation and Routing

The use of React Router ('BrowserRouter', 'Routes', 'Route', 'Link') provides client-side routing, enabling users to navigate from page to page. This can be seen in App.js.

Backend Design

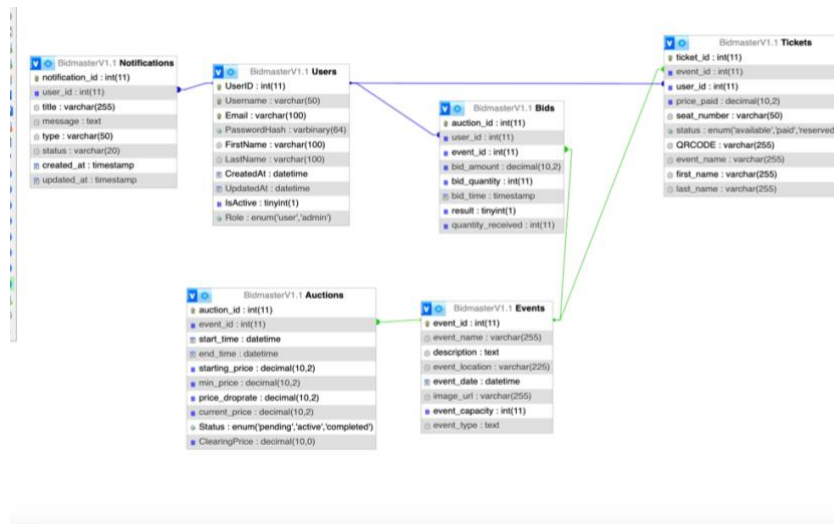
Technologies Used

- **Node.js & Express:** Forms the server, enabling the creation of RESTful API endpoints for data retrieval and manipulation.
- **MySQL:** The MySQL database stores and organizes all user, event, bid, auction, and ticket data.
- **Bcrypt:** Secures passwords through hashing.
- **JWT (JSON Web Tokens):** Facilitates user authentication, securing a session for each user after they log in.
- **CORS (Cross-Origin Resource Sharing):** Allows the application to safely and securely access resources from the server across different domains.
- **Node-cron:** Schedules tasks to run at specific times. (Such as checking for finished auctions)

Database Structure

The MySQL database includes several interconnected tables with key relations:

- **Users:** Stores User's credentials and details and encrypted Passwords.
- **Events:** Stores all Event Data and Details.
- **Auctions:** Linked with Events it holds auction details such as time and pricing information.
- **Bids:** Records all Bids placed by Users on Events.
- **Tickets:** Stores all the Tickets Generated for Winners.
- **Notifications:** Stores all notifications that are sent and read by Users.



REST API Endpoints

The backend has various endpoints for operations such as User Registration ('/register'), User Sign-in ('/signin'), Event Management ('/events/:event_id'), Bidding on events ('/events/:event_id/bid') and User Notifications ('/user/:UserID/notifications'). Each endpoint executes specific database queries to retrieve send or manipulate data.

- **User Authentication:** Includes registering and sign-in logic with password hashing and JWT token generation.

```

//REGISTER USERS
app.post('/register', (req, res) => {
  const { email, password, firstname, lastname } = req.body;
  let { username } = req.body;
  //IF USERNAME NOT PROVIDED SET IT AS EMAIL
  username = username || email;
  //BCRYPT HASHING FOR PASSWORD SECURITY
  bcrypt.hash(password, saltRounds, (err, hash) => {
    if (err) {
      console.error(err);
      return res.status(500).json({ message: "Error hashing password" });
    }
    //SQL INSERT STATEMENT FOR INSERTING NEW USERS INTO TABLE
    const sql = "INSERT INTO Users (Username, Email, FirstName, LastName, PasswordHash) VALUES (?, ?, ?, ?, ?)";
    db.query(sql, [username, email, firstname, lastname, hash], (error, results) => {
      if (error) {
        console.error(error);
        //IF THERE IS ALREADY A USER ASSIGNED WITH DETAILS THROW ERROR
        if (error.code === 'ER_DUP_ENTRY') {
          return res.status(409).json({ message: "Email or Username already exists" });
        }
        return res.status(500).json({ message: "Error registering new user" });
      }
      res.status(201).json({ message: "User registered successfully", userId: results.insertId });
    });
  });
});

//USER SIGN IN
app.post('/signin', (req, res) => {
  const { email, password } = req.body;
  //CHECK DATABASE FOR USER WITH CORRESPONDING EMAIL
  const sql = "SELECT * FROM Users WHERE Email = ?";
  db.query(sql, [email], (err, users) => {
    if (err) {
      console.error(err);
      return res.status(500).json({ message: "Server error" });
    }
    //IF NOT USERS RETURNED
    if (users.length === 0) {
      return res.status(404).json({ message: "User not found" });
    }
    const user = users[0];
    //AUTHENTICATE PASSWORD
    bcrypt.compare(password, user.PasswordHash.toString(), (err, isMatch) => {
      if (err) {
        console.error(err);
        return res.status(500).json({ message: "Server Error during password authentication" });
      }
      //IF PASSWORD DOESNT MATCH
      if (!isMatch) {
        return res.status(401).json({ message: "Incorrect Password" });
      }
      const id = user.id;
      //CREATE JWT TOKEN
      const token = jwt.sign(id, 'jwtSecret', {
        //EXPIRY TIME
        expiresIn: 360, //360S
      });
      //CORRECT PASSWORD
      return res.json({ message: "Correct Password", users: { UserID: user.UserID, Username: user.Username, Email: user.Email, FirstName: user.FirstName, LastName: user.LastName } });
    });
  });
});
  
```

- **Event Management:** CRUD operations are used for creating, updating and deleting events, using transactions to handle related auction data such as placing user's bids.

```

//ADMIN SETTINGS
//ADD NEW EVENT ALONG WITH ITS AUCTION
app.post('/addevent', (req, res) => {
  const { eventName, eventDesc, eventDate, eventLocation, eventURL, eventCapacity, startingPrice, minimumPrice, priceDropRate, auctionStartTime, auctionEndTime, event }
  //ENSURE ALL DATA IS SENT
  if (!eventName || !eventDesc || !eventDate || !eventLocation || !eventURL || !eventCapacity || !minimumPrice || !auctionStartTime || !auctionEndTime || !eventType)
    return res.status(400).json({ message: "Missing required fields" });
  //INSERT NEW EVENT DATA
  const eventsSql = "INSERT INTO Events (event_name, description, event_location, event_date, image_url, event_capacity,event_type) VALUES (?, ?, ?, ?, ?, ?, ?)";
  db.query(eventsSql, [eventName, eventDesc, eventLocation, eventDate, eventURL, eventCapacity,eventType], (eventError, eventResults) => {
    if (eventError) {
      console.error(eventError);
      return res.status(500).json({ message: "Error adding event" });
    }
    //SET EVENTID TO LAST INSERTED ID IN EVENT TABLE
    const eventId = eventResults.insertId;
    //INSERT AUCTION DATA CORRESPONDING TO EVENT DATA
    const auctionsSql = "INSERT INTO Auctions (event_id, min_price,start_time, end_time) VALUES (?, ?, ?, ?)";
    db.query(auctionsSql, [eventId,minimumPrice, auctionStartTime, auctionEndTime], (auctionError, auctionResults) => {
      if (auctionError) {
        console.error(auctionError);
        return res.status(500).json({ message: "Error adding auction data" });
      }
      res.status(201).json({ message: "Event and Auction added successfully", eventId: eventId, auctionId: auctionResults.insertId });
    });
  });
});

```

- **Bid Placement:** Handles the Logic for users placing bids and ensures validation.

```

//PLACE NEW BID
app.post('/events/:event_id/bid', (req, res) => {
  //GET VALUES FROM THE REQUEST BODY
  const { UserID, bidAmount, quantity } = req.body;
  //GET EVENT ID FROM URL
  const { event_id } = req.params;
  //INSERT NEW BIDS INTO TABLE
  const sql = "INSERT INTO Bids (user_id, event_id, bid_amount, bid_quantity) VALUES (?, ?, ?, ?)";
  db.query(sql, [UserID, event_id, bidAmount, quantity], (error, results) => {
    if (error) {
      console.error(error);
      return res.status(500).json({ message: "Error adding bid" });
    }
    res.status(201).json({ message: "Bid added successfully", bidId: results.insertId });
  });
});

```

- **Scheduled Tasks:** Node-cron is used to periodically check for completed auctions and executes the bid processing logic, determining winning and losing Bids and generating Tickets for successful Bids.

```

//NODE CRON FOR SCHEDULING EVENTS
const cron = require('node-cron');

//NODE-CRON SCHEDULER FOR CHECKING FOR ENDED AUCTIONS AND PROCESSING THEM
cron.schedule('* * * * *', ()=>{
  checkAuctionStatus();
})

```

Key Backend Functions

Process Bids()

The most important part of the whole Backend is the 'processingBids()' function that is invoked at the end of each auction. It follows a series of key steps to sort bids, determine the winners based on the availability of tickets and their bid amounts, set Final ticket prices and update Auction Status. The result of all these tasks defines a Blind Dutch Auction.

Step-by-Step Functionality

1. **Fetching and Sorting Bids:** The function begins by retrieving all bids placed for a given event, sorting them in descending order by bid amount and in ascending order by the time the bid was placed (In the events of Ties the earlier bid wins).
2. **Determining Available Tickets:** It then fetches the total amount of tickets available.
3. **Allocating Tickets:** The function iterates over the sorted bids (from top to bottom), allocating tickets to each bid until all tickets are distributed or until bids run out. It follows these rules:
 - If a bid requests more tickets than available. It will receive only the number that's available.
 - If there is a tiebreaker the earlier bid wins.
 - The Final Ticket Price is determined by the last bid that receives.
4. **Updating Bid Result:** After all tickets are distributed, each bid is updated to reflect its outcome:
 - Winning Bids are marked in the Bids Table with the result set to 1 and the quantity received updated accordingly.
 - Losing Bids are marked with the result 0.
5. **Generating Tickets:** For each winning bid the 'generateTickets()' function is called to create tickets and sends them to the Tickets table.
6. **Notifications:** Notifications are sent to All users who placed bids informing their bid result.
7. **Updates Auction Status:** Auction Status is set from 'in progress' to 'complete'.

Error Handling and Transactions

The function is designed to handle errors at each step, ensuring that if any part of the process fails, the entire process can be rolled back to maintain data consistency. This is very important when it comes to dealing with financial transactions which will be implemented in future versions.

Scheduled Processing

The function is scheduled to run automatically at the end of an auction using 'node-cron'. This ensures the timely processing of bids and doesn't require manual implementation.

GenerateTickets()

The 'generateTickets()' function ensures that each winner is allocated the correct number of tickets. These tickets are recorded in the 'Tickets' table, complete with unique details for each ticket, such as ticket ID, seat number and QRcode.

Step-by-Step Functionality

1. **Iteration over Winning Bids:** The function iterates over the array of winning bids, which includes 'user_id', 'bid_amount' and 'quantity_recieved' for each bid.

2. **Ticket Generator:** For each bid, it generates the required number of tickets as specified by 'quantity_recieved'. It ensures that each ticket has a unique seat number and reference number.
3. **Database Insertion:** Insert each generated ticket and its details into the 'Tickets' table.
4. **Looping for Multiple Tickets:** If a user has won multiple tickets, the function creates individual entries for each ticket, ensuring each ticket has a different seat number and reference code.
5. **Error Handling:** It includes error handling to ensure that if the generation process runs into any issues, appropriate error messages are logged and transactions can be rolled back to maintain database integrity.
6. **Notifications:** After successfully generating tickets, the function calls the 'sendNotification()' function, to inform users that their tickets are ready.

CheckAuctionStatus()

The purpose of the 'checkAuctionStatus()' function is to look for auctions that have been ended and not yet been processed and call the processBids() function. This function is run every minute using the node.cron scheduler.

```
//CHECK AUCTION STATUS
async function checkAuctionStatus(){
  console.log('Checking for Ended Auctions and processing them.....');
  try{
    //GET ENDED AUCTIONS
    const endedAuctions = await queryPromise('SELECT * FROM Auctions WHERE end_time <= NOW() AND Status != "completed"');
    //ITERATE THROUGH ENDED AUCTIONS
    for(const auction of endedAuctions){
      //CALL PROCESS BIDS FUNCTION
      await processBids(auction.event_id);
      //UPDATE STATUS TO COMPLETED
      await queryPromise('UPDATE Auctions SET Status = "completed" WHERE event_id = ?', [auction.event_id]);
    }
    console.log('Finished Checking');
  }catch(error){
    console.error('Error Checking and processing ended auctions: ',error);
  }
}

//NODE-CRON SCHEDULER FOR CHECKING FOR ENDED AUCTIONS AND PROCESSING THEM
cron.schedule('* * * * *',()=>{
  checkAuctionStatus();
})
```

FAQs

- **Q: How Does the Blind Dutch Auction Work on Bidster?**
Ans: The Blind Dutch Auction on Bidster allows you to bid the maximum amount you're willing to pay per ticket without knowing the bids of other participants. Once the auction is over, winners all pay the same price, the Clearing Price, which is the lowest winning bid, ensuring fairness and market-driven pricing.
- **Q: How can I place a Bid?**
Ans: To place a bid, first ensure you're logged in. Navigate to the event you're interested in and enter your maximum bid amount and the number of tickets you wish to purchase. Your bid will be processed once the auction closes, and you'll be sent a notification of the outcome.
- **Q: What happens if my bid wins?**
Ans: If your bid is successful, the bid amount will be charged to your payment method, and your tickets will be available under "My Tickets" in your account.

- **Q: Can I modify or cancel a bid?**
Ans: Once a bid is placed, it cannot be cancelled or modified.
- **Q: What happens if my bid loses?**
Ans: If your bid loses, you will not be charged. You can attempt to bid on other events or future auctions for the same event if available.
- **Q: How are Auction closing times determined?**
Ans: Auction closing times are set by the event organizer and can vary between events. Each event page displays a countdown timer indicating when the auction will close.
- **Q: I'd like to allow users to purchase tickets immediately bypassing the auction process. How can I set this up?**
Ans: This facility is not yet available, but may be implemented in a future version.