

DISTRIBUTED SYSTEMS

FINAL PROJECT

ART AUCTION HOUSE

APPLICATION/API

ZHYAR YASSIN ABDALLA

1 Artwork Auction House

- **Description:** The art auction system is a web-based platform that allows users to buy and sell artwork through online auctions. Users can browse, bid on, and purchase art pieces. Artists can register and list their art for auctions, and users can participate in auctions by placing bids.
- **Purpose:** The purpose of this application is to make an online art auction that helps artists sell their products in a trustworthy online auction and to build an API that can be shared among different applications to help sell artworks at a good price so many people will be aware of each artwork auction.
- **Scope:** The proposed API will include the following key features:
 - Real-Time Bid Updates: Using a real-time bid update mechanism, users can receive instant notifications on bid status, competing offers, and auction progress.
 - Integration with External Art Databases and Catalogs: Integrating with external art databases and catalogs to provide users with a diverse and expansive collection, thereby increasing the variety of artworks available for auction,
 - User Authentication and Authorization: Implementing a secure authentication system to protect user information and ensure authorized application access
- **Impact:** The following advantages are expected to result from the implementation of this custom API:
 - Improved User Experience: Real-time bid updates and access to a broader range of artworks will increase user engagement and satisfaction significantly.

- Increased Auction Participation: Simplified processes, secure transactions, and a diverse art collection will draw a larger audience, resulting in increased auction participation.
- Scalability: Due to the modular design of the API, future expansion and integration of additional features will be possible, ensuring the scalability of our art auction application.

2. API Design

- **API Endpoints:**

- **USER: Endpoint: 'api/user'**

	Route	Method	Function
1	'api/user'	GET	Get all users
2	'api/user'	POST	Create user
3	'api/user/:id'	GET	Get a single user
4	'api/user/:id'	PUT	Update user
5	'api/user/:id'	DELETE	Remove user
6	'api/user/login'	POST	User Authentication & Authorization
7	'api/user/profile'	GET	Check user auth
8	'api/user/logout'	POST	Destroy token

- **ARTWORK: Endpoint: 'api/artwork'**

	Route	Method	Function
1	'api/artwork'	GET	Get all artworks
2	'api/artwork'	POST	Create artwork
3	'api/artwork'	GET	Get a single artwork
4	'api/artwork/:id'	PUT	Update artwork
5	'api/artwork/:id'	DELETE	Remove artwork
6	'api/artwork/by-bidder/:last_bidder'	GET	Get artwork based on last user bid
7	'api/artwork/by-bidder/:artist_id'	GET	Get artwork based on artist

- **ARTIST: Endpoint: '/artist'**

	Route	Method	Function
1	'api/artist'	GET	Get all artists
2	'api/artist'	POST	Create artist
2	'api/artist'	GET	Get a single artist
3	'api/artist/:id'	PUT	Update artist
4	'api/artist/:id'	DELETE	Remove artist
5	'api/artist/user/:id'	GET	Get artist based on user id

- **ARTIST: Endpoint: 'api/upload'**

	Route	Method	Function
1	'api/upload/uploadimage/:id'	POST	Add profile image to artist

- **API Endpoints:**

- **USER:**

```
▪  username: {
▪    type: String,
▪    unique: true,
▪    required: true,
▪  },
▪  password: {
▪    type: String,
▪    required: true,
▪  },
▪  email: {
▪    type: String,
▪    unique: true,
▪    required: true,
▪  },
▪  user_type: {
▪    type: String,
▪    enum: ["artist", "bidder"],
▪    required: true
▪  },
▪  profile_image: {
▪    type: String,
▪  }
}
```

- **ARTWORK:**

```
▪  title: {
▪    type: String,
▪    required: true,
▪  },
▪  description: {
▪    type: String,
▪  },
▪  artwork_image: {
▪    type: String,
```

```

    },
    artist_id: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Artist',
      required: true,
    },
    starting_bid: {
      type: Number,
      required: true,
    },
    startTime: {
      type: Date,
      required: true,
    },
    endTime: {
      type: Date,
      required: true,
    },
    currentHighestBid: {
      type: Number,
      required: true,
    },
    last_bidder: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User',
    },
  },

```

○ ARTIST:

```

    user_id: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User',
      unique: true,
      required: true,
    },
    bio: {
      type: String,
    },
  },

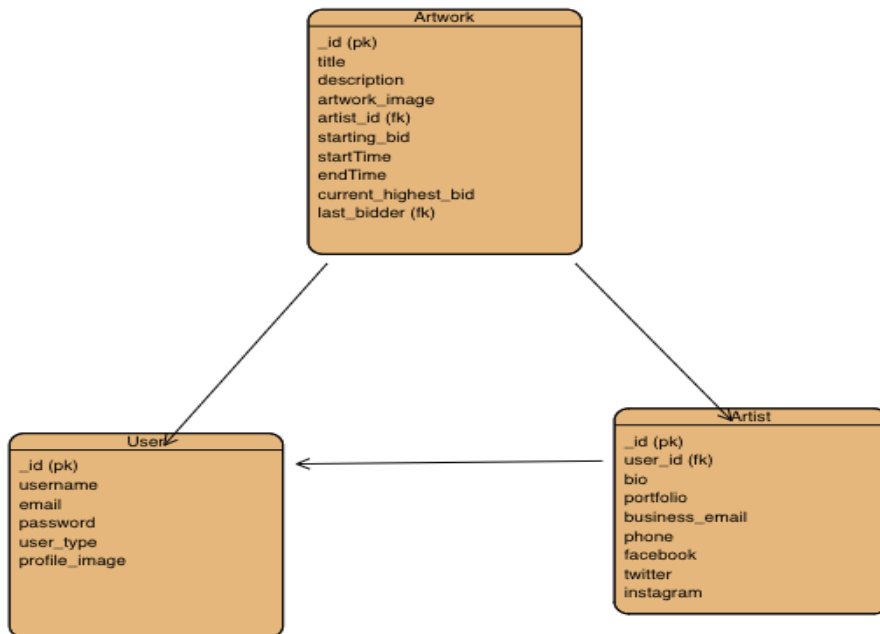
```

```
▪ portfolio: {  
▪   type: String,  
▪ },  
▪ business_email: {  
▪   type: String,  
▪   required: true,  
▪ },  
▪ phone: {  
▪   type: String,  
▪ },  
▪ facebook: {  
▪   type: String,  
▪ },  
▪ twitter: {  
▪   type: String,  
▪ },  
▪ instagram: {  
▪   type: String,  
▪ },
```

3. Data Management and Storage

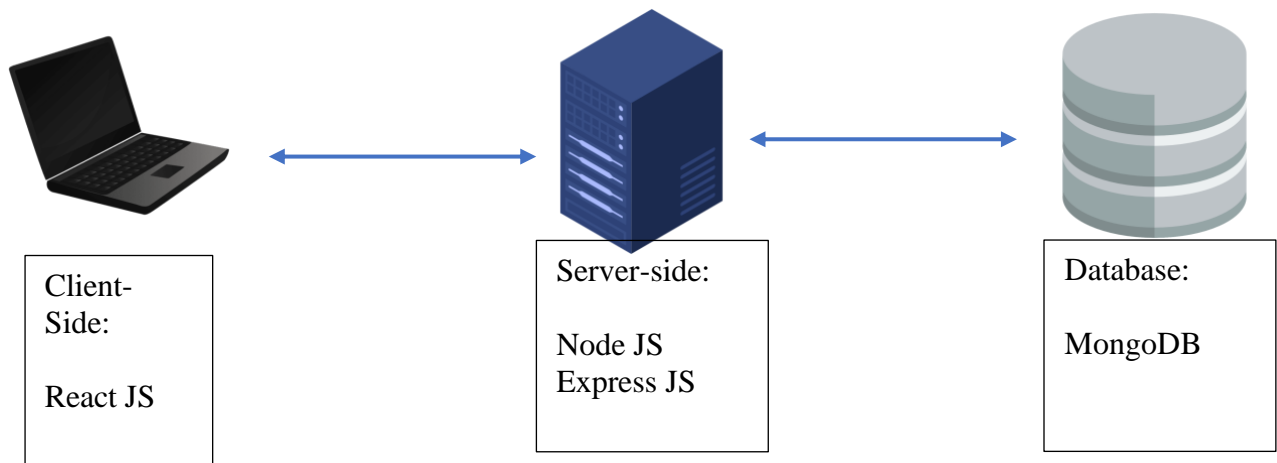
MongoDB is used as a non-relational (NoSQL) database management system, Cloud version of MongoDB is used so from everywhere around the world can be accessed and also the main idea is to use in localhost from different users to have same data while development.

MongoDB database design follows as:



4. Application Architecture

- **Application Structure:** The system is implemented using a client-server architecture. The server handles API requests and communicates with a database for storing artwork, user, and auction data. The client is a web application built with React for the frontend and Express.js for the backend.



- **Distributed system features and algorithms:**

- **Client-Server Architecture:**

- **Client Application:** the client application is implemented using one of the most popular frameworks which it is React JS along with PrimeReact library for using prebuilt components.
 - **Server Application:** Express JS is used as a framework for Node JS to make the API with other techniques such as (Tokens, Encryption, Web-sockets, etc.).
 - **Database:** MongoDB is used as a non-relational database to store data.

- **Real-Time Message Exchanging:** using web-sockets helps to make a real-time connection between multiple users on a single artwork auction which enables users to connect to same auction and start bidding in real-time execution.

- **API:** the entire back-end of the system is built with API which various systems can be utilized based on this API implementation.

- **Testing Plan:** The system will be tested at different levels, including unit testing for individual components, integration testing for server-client interactions, and end-to-end testing for user scenarios.
- **System Tiers:** The system consists of three tiers: the presentation tier (frontend built with React), the application tier (Express.js server handling business logic), and the data tier (MongoDB database for storing art, user, and auction data).

5. Application UI & Features

- **Users can login or Register**

LOGIN

Email

Password

DONE

Register

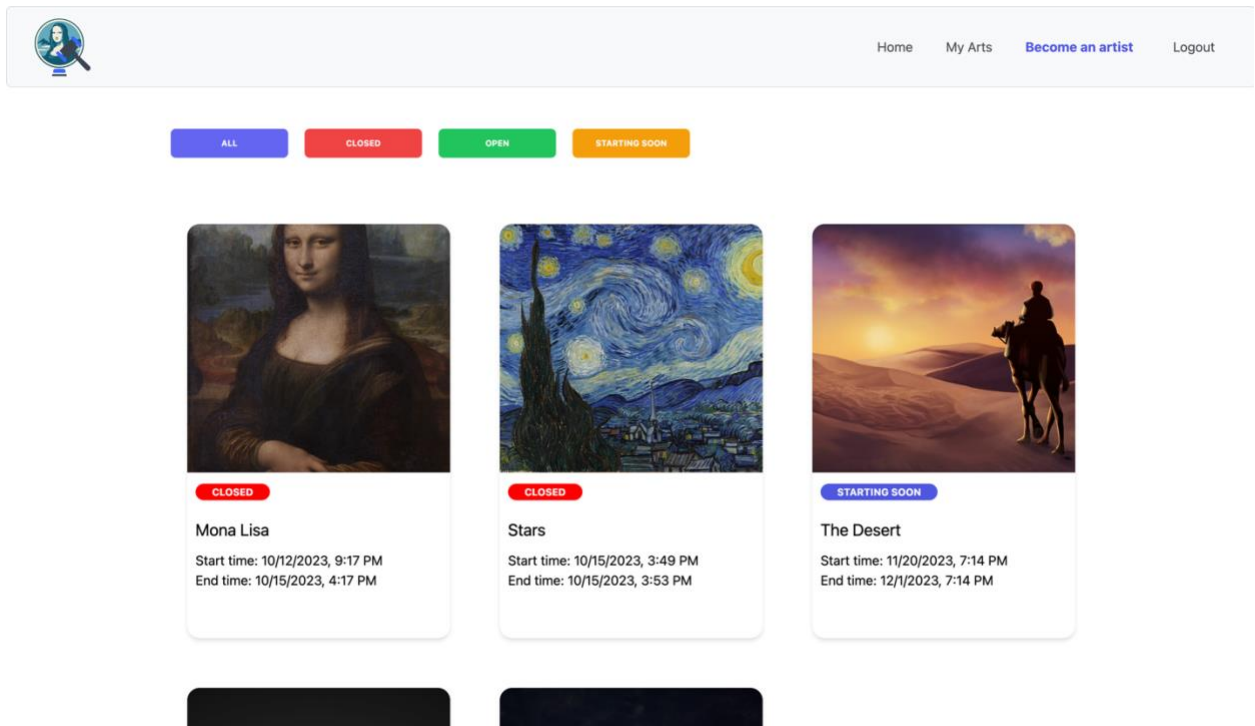
Username

Email

Password

DONE

- **Users can view artworks**



- **Users can become an artist to sell artworks**

The screenshot shows the 'Register as an artist' form. At the top, there is a navigation bar with a profile icon, a search icon, and links for Home, My Arts, Become an artist, and Logout. The form is titled 'Register as an artist' and contains several input fields: 'Bio', 'Portfolio', 'Business Email', and 'Business Phone'. Below these fields, there is a file upload section with a '+ Choose' button, an 'Upload' button, and a 'Cancel' button. Below the upload buttons, there is a text area with the instruction 'Drag and drop files to here to upload.'.

- **Artists can add artwork for auction.**

Add an artwork for auction

Title

Description

0


Start date

End date

+ Select File


DONE

- **Users can bid on arts.**



THE MOUNTAIN

qwe


zhyar yassin abdalla

🕒 11/19/2023, 7:52 PM
🕒 12/15/2023, 7:52 PM

Time remain to bid: 618:27:16

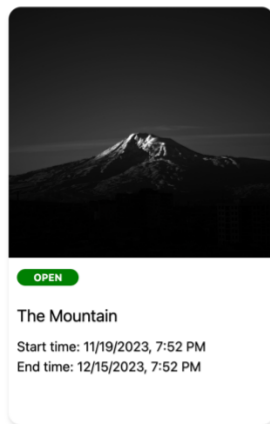
30000

618:27:16

0

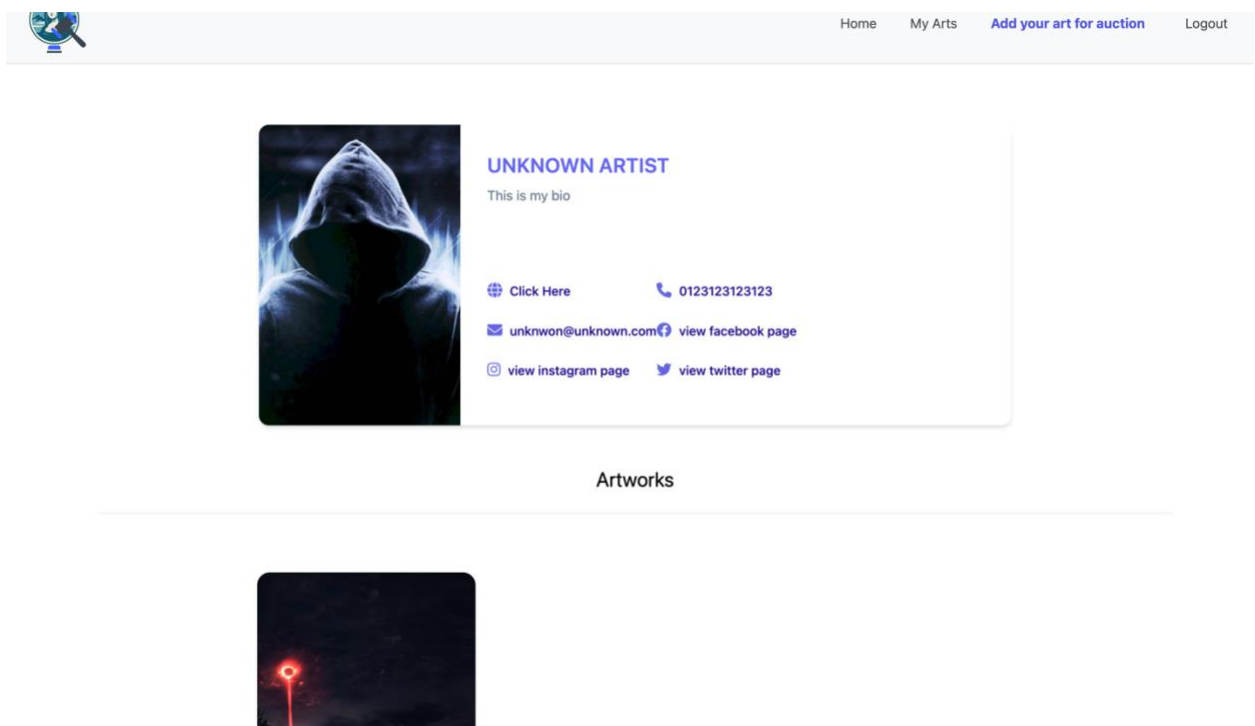
Bid


- **Users can see the arts they might win or already won in the auction**



https://art-auction-house.xyz/my/auctions#


- **Users can see artists’ profile and their auctions or arts**








UNKNOWN ARTIST


This is my bio



[Click Here](#)


[0123123123123](#)


unknwon@unknown.com


[view facebook page](#)


[view instagram page](#)


[view twitter page](#)

Artworks



- **Users can filter auction based in their status**

[Home](#)[My Arts](#)[Add your art for auction](#)[Logout](#)[ALL](#)[CLOSED](#)[OPEN](#)[STARTING SOON](#)[CLOSED](#)

Mona Lisa

Start time: 10/12/2023, 9:17 PM
End time: 10/15/2023, 4:17 PM

[CLOSED](#)

Stars

Start time: 10/15/2023, 3:49 PM
End time: 10/15/2023, 3:53 PM

Documentation for Design and Implementation:

Art Auction Project Documentation

Introduction

The art auction system is a web-based platform that allows users to buy and sell artwork through online auctions. Users can browse, bid on, and purchase art pieces. Artists can register and list their art for auctions, and users can participate in auctions by placing bids.

System Architecture

- Client-side (Frontend) architecture
- Server-side (Backend) architecture
- Communication between components

API Specification

#Routes

```
app.use('/api/user', userRoute);
app.use('/api/artist', artistRoute);
app.use('/api/artwork', artworkRoute);
app.use('/api/upload', uploadFileRoute);
app.use('/uploads', express.static('uploads'));
```

Request and response Format: the API uses all HTTP request methods (GET, POST, PUT, DELETE).


```

// Fetch all artists
router.get('/', async (req, res) => {
  try {
    const artists = await Artist.find();
    res.json(artists);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// Update artist by ID
router.put('/:id', async (req, res) => {
  try {
    const updatedArtist = await Artist.findByIdAndUpdate(
      req.params.id,
      req.body,
      { new: true }
    );
    if (!updatedArtist) {
      return res.status(404).json({ message: 'Artist not found' });
    }
    res.json(updatedArtist);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// Delete artist by ID
router.delete('/:id', async (req, res) => {
  try {
    const deletedArtist = await Artist.findByIdAndRemove(req.params.id);
    if (!deletedArtist) {
      return res.status(404).json({ message: 'Artist not found' });
    }
    res.json({ message: 'Artist deleted' });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// Create a new artist
router.post('/', async (req, res) => {
  try {
    const newArtist = new Artist(req.body); // req.body should contain artist data
    console.log(newArtist);
    const savedArtist = await newArtist.save();
    console.log(savedArtist.user_id);
    const updatedUser = await User.findByIdAndUpdate(savedArtist.user_id, { user_type: 'artist' }, { new: true });

    if (!updatedUser) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.status(201).json(savedArtist);
  } catch (err) {
    console.log(err.message);
    res.status(400).json({ error: err.message });
  }
});

```

Authentication Methods:

```

router.post('/login', async (req, res) => {
  try {
    const { email, password } = req.body;
    const userDoc = await User.findOne({ email });

    if (!userDoc) {
      return res.status(404).json({ message: 'User not found' });
    }

    const passOk = bcrypt.compareSync(password, userDoc.password);
    if (passOk) {
      jwt.sign(
        { email: userDoc.email, user_type: userDoc.user_type, username: userDoc.username, id: userDoc._id },
        jwtSecret,
        {},
        (err, token) => {
          if (err) throw err;
          res.cookie('token', token).json(true);
        }
      );
    } else {
      res.status(422).json('Password not correct');
    }
  } catch (err) {
    console.error('Error:', err);
    if (err instanceof mongoose.Error) {
      return res.status(500).json({ error: 'Database error' });
    }
    res.status(400).json({ error: err.message });
  }
});

```

```

router.get('/profile', (req, res) => {
  const token = req.cookies['token'];
  if(token){
    jwt.verify(token, jwtSecret, {}, async (err, userData) => {
      console.log("userData", userData);
      if(err) throw err;
      const user = await User.findById(userData.id);
      res.json(user);
    });
  }else{
    res.json(null)
  }
});

```

Frontend and Backend Implementation

1. Frontend (Client-side)

Overview:

The frontend of our art auction application is built using React, a popular JavaScript library for building user interfaces. It provides a user-friendly interface for users to interact with the system.

Key Components:

- **React Components:** Our frontend is organized into components, each responsible for specific user interface elements and functionality.
- **State Management:** We use React's state and context API for managing the application's state, including user authentication and data retrieval.
- **Routing:** We implement client-side routing using React Router to enable navigation between different pages and views.

Implementation Highlights:

- **User Registration and Login:** We provide user registration and login features with form validation and authentication.
- **Auction Listings:** Users can browse and search for artwork listings. We use Axios to make API requests to retrieve auction data.
- **Real-time Bidding:** We implement real-time bidding using WebSockets (Socket.io). Users can participate in auctions and receive live updates on bidding activities.

2. Backend (Server-side)

Overview:

The backend of our art auction application is built using Express.js, a Node.js framework for building web applications and APIs. It handles data storage, business logic, and API requests.

Key Components:

- Express Routes: We define various routes and controllers to handle API requests related to users, auctions, artwork, and bids.
- Database Interaction: We use MongoDB as our database system to store user data, auction information, artwork details, and bid history.
- WebSockets: Socket.io is used for real-time communication between clients and the server for live bidding updates.

Implementation Highlights:

- User Authentication: We implement user authentication using JWT (JSON Web Tokens). Users can log in, and authenticated routes are protected.
- Auction Management: The backend handles the creation, updating, and closing of auctions, as well as tracking bids and highest bidders.
- WebSocket Integration: Socket.io enables real-time communication for bidding and auction updates. It broadcasts events to connected clients.

Testing Methodology

- Unit Testing: We conduct unit tests for individual components, such as functions, modules, and React components. These tests ensure that each component works as expected in isolation.
- Integration Testing: Integration tests focus on interactions between different parts of the system. We test how various components work together to ensure seamless functionality.
- End-to-End Testing: End-to-end tests simulate user scenarios and workflows to validate the entire system's functionality. These tests cover user journeys from start to finish.

Deployment

I successfully deployed my art auction website on Regxa's shared web hosting platform using cPanel. Here's a step-by-step guide on how I accomplished this:

Server Setup

- Buying Shared Host: I bought a shared web host from Regxa which and also a domain for the website.
- Accessing cPanel: After signing up with Regxa, I received my cPanel login credentials. I used these credentials to log in to my cPanel account, which is the control panel for managing the hosting environment.
- Domain Configuration: I ensured that my primary domain or subdomain was configured correctly to point to my Regxa hosting account. This step is essential to make my website accessible via the domain name.
- File Upload: In cPanel, I navigated to the "File Manager" tool. Here, I uploaded my art auction website files to the hosting server. For the primary domain, I uploaded the files to the `public_html` directory. In the case of a subdomain, I created a subfolder within `public_html` for that subdomain and uploaded the files there.