Auctionly

an online art auctioning experience

Nutsa Chichilidze - 18131956
Ranya El-Hwigi - 18227449
John O'Keeffe - 18249531
Erona Aliu - 18228402

Table of Contents

Narrative description of business scenario

Auctionly is a web-based application which offers an exhaustive art auctioning experience without the need of leaving your home. It caters to people interested in selling or buying digital art pieces online.

We allow artists to exhibit their work through our website, which then reaches our customer base. Artists have the ability to create a profile on our website, and describe personal details about themselves and their art to allow buyers to get to know them. They have the power to publish their art, fully customise their auction properties - or pick one of the auction types designed by us. We rank our sellers by popularity and offer commission discounts to our most desirable artists. We also keep close track of the auction progress and provide suggestions if an art piece is not performing well during the auction.

Buyers are also able to create personal accounts on our website, where they can specify the type of art they are interested in. They are allowed to post bids on auctions and purchase art pieces when they win. We encourage our buyers by displaying a collection of their art pieces under their profile. We also rank our customers by activity and offer special discounts to our most loyal and determined auctioneers.

The software lifecycle model adopted

The task of choosing a suitable software lifecycle is a challenging one. We want one that will allow us to have a clear understanding between team representatives about *when* we should do *what* to avoid setbacks and miscommunication amongst the team as well as splitting work fairly between the team.

The main SDLC Models we looked at were:

1. Waterfall Model
2. RAD (Rapid Application Development) Model
3. Agile Model

Model Name	Advantages	Disadvantages	Suitable Projects
Waterfall	• Phase dependency on the one before it. • Smaller projects where requirements are defined. • Tests before each phase is finished.	• Errors only be fixed during given phase. • Not good for large complex projects. • Documentation requires a lot of time. • Client feedback not included during development phases.	• Requirements not changing often. • Short Project. • Readily available resources.
RAD	• Adapt to change. • Less people, increased productivity. • Code generation - save time.	• Not all applications are RAD compatible. • High-skilled developers required. •	• Short Project • Requirements known • Less tech risks • High budget (modelling + automated tools)
Agile	• Deploy software quickly and get constant customer feedback. • Fast turnaround times. • Immediate feedback. • Experimentation and tests - low costs. • Increased focus on customer needs.	• Documentation sidetracking. • Several cycles - difficult to keep track of progress. • No clear end. • Short cycle - can feel quite rushed.	• Small and medium sized software development. • Main deliverable can follow divide and conquer strategy.

With this in mind, after a round of research, as well as some industry experience between our team members, the suitable lifecycle method the team agreed on was the Agile Model.

Project Plan

Deliverable	Description	Team Member	Week
Set up team roles	Allocating specific roles to each member and ensuring understanding of responsibilities.	All Members	3
Research existing projects and agree on scenario	Brainstorming on project ideas that are interesting to all team members and viable within the given time frame.	All Members	3
Describe the business scenario	Document the business scenario as a part of the project documentation.	Nutsa	3
Project plan	Devise a project plan and fill in the project documentation appropriately.	Nutsa	3
Choose a software lifecycle model	Researching a variety of software lifecycle models and agreeing with the team on a specific one. Producing a report for the project documentation.	Erona	3
Requirements Analysis	Produce a list of functional and non-functional requirements. Describe tactics to support quality attributes.	Ranya	4
Use case analysis	Produce a list of use cases. Create a use case diagram. Allocate requirements to use cases.	Ranya	4
Sample GUI prototypes	Generate prototypes of the software product. Brainstorm on brand colours/feel/style.	Johnny	4
Research technical tools for the project	Compare technical tools and pick the appropriate ones for the project. Create a DevOps environment (i.e. GitHub repository)	Johnny	5
Analysis sketches	Sketch of the analysis class diagram, with significant methods and attributes identified. Demonstrate inheritance, aggregation, composition, associations, dependencies, visibility, etc.	Erona	5
List of candidate objects	Candidate objects derived using Data Driven Design	Erona	5
Communication diagram/sequence diagram	Provide a sketch of either for the project document	Erona	5
Entity relationship diagram with cardinality		Erona	5

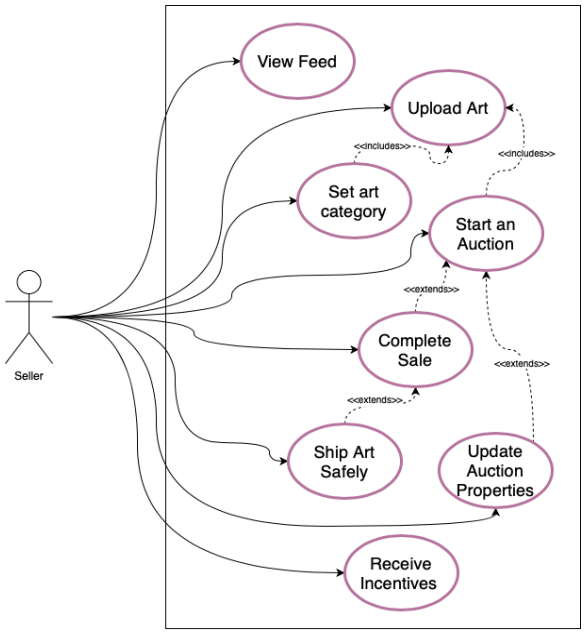
High level architecture design		Nutsa	6
Code: a basic infrastructure	Build basic infrastructure using Python.	All Members	7
Code: use case #1, use case #2	Write code for 2 use cases.	All Members	7
Code: use case #3, use case #4	Write code for 2 use cases.	All Members	8
Code: use case #5	Write code for a fifth use case.	All Members	9
Code: GUI	Write code for the user interface.	All Members	
Code: added value	Use of concepts not covered in the lectures, for example: Object Relational Mapping (ORM) REST architectural pattern Concurrency through threading or OpenMP. Security Software Metrics DevOps	All Members	9
Illustrations of added value	Coding fragments and/or screen shots with brief descriptions in this section of the report to illustrate implementation of Added Value.	All Members	9
Architecture and design recovery	Based on the implementation but can also specify concepts that could be developed in future releases. Draw blueprints of: a. Architectural diagram b. Design-time class diagram. c. One state chart for an object in either the sequence or communication diagram, with annotated transition strings.	Ranya	10
Critique	Compare and contrast the analysis sketches in (8) versus the blueprints created in (12).	Ranya and Erona	10
Listing of package structure	A listing in tabular format of classes in each package, their authors, and lines of code. Also include total lines of code developed (a) for the application as a whole, and (b) per team member.	All Members	10
References	Collect a list of references used for the project.	All Members	10

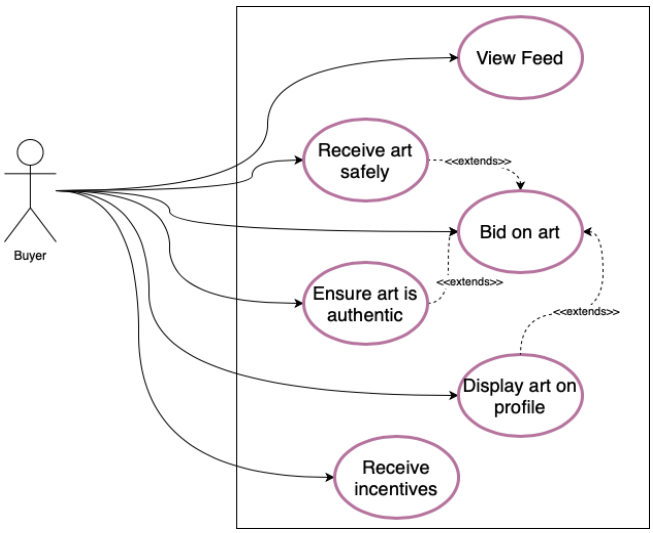
Project Roles

Role	Description	Team Member
Project Manager	Sets up group meetings, gets agreements on project plan, tracks progress	Ranya El-Hwigi
Documentation Manager	Sources relevant documentation from each team member and composes it in the report	Erona Aliu
Business Analyst/Requirements Engineer	Responsible for section 6: requirements	Ranya El-Hwigi
Architect	Defines system architecture	Nutsa Chichilidze
Systems Analyst	Creates conceptual class model	Erona Aliu
Designer	Responsible for recovering design time blueprints from all implementation	Ranya El-Hwigi
Technical Lead	Leads the implementation effort	John O'Keeffe
Programmer(s)	Develops at least 1 package of architecture	All Members
Tester	Codes automated test cases	Nutsa Chichilidze
Dev Ops	Ensures all team members are competent with GitHub, Bamboo, etc.	John O'Keeffe

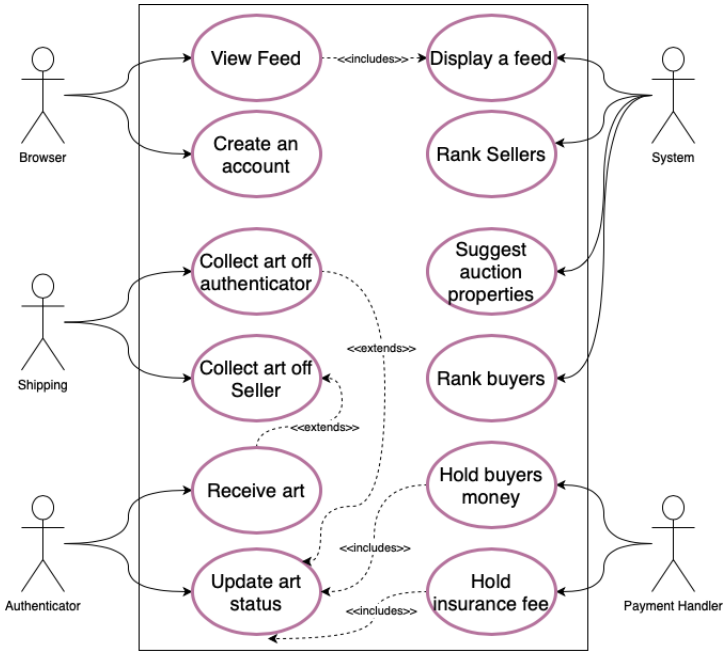
Requirements

Use cases





1. Seller
    - 1.1. As a seller I want to be able to upload my art so that I can later put it up for exhibition or auction.
    - 1.2. As a seller I want to be able to complete a sale of my art so I can earn money.
    - 1.3. As a seller I want to be able to update my ongoing auctions properties so that I can help it do better to increase my chances of a higher bid.
    - 1.4. As a seller I want to be able to view the feed so that I can see the art available.
    - 1.5. As a seller I want to be able to receive incentives for my loyalty so that I have added benefits.
    - 1.6. As a seller I want to be able to ship the art safely so that it doesn't get damaged.
    - 1.7. As a seller I want to be able to set the art category my art relates to so that it reaches the correct buyers.
    - 1.8. As a seller I want to be able to start an auction for my art so that buyers can bid on it.
2. Buyer
    - 2.1. As a buyer I want to have the art I buy displayed on my profile so that other users can see.
    - 2.2. As a buyer I want to be able to view a feed of the art so that I know what's available.
    - 2.3. As a buyer I want to be able to bid on art in auctions so that I can buy them.
    - 2.4. As a buyer I want to be sure the art I'm receiving is authentic so that I don't get scammed.
    - 2.5. As a buyer I want to have the art I buy shipped safely so that it doesn't get damaged.
    - 2.6. As a buyer I want to be able to receive incentives for my loyalty so that I have added benefits.



3. Browser
    3.1. As a browser I want to be able to view the art that is available on the website so that I don't have to create an account unless I'm interested.
    3.2. As a browser I want to be able to create an account if I want to so that I can buy/sell art.
4. Authenticator
    4.1. As an authenticator I want to be able to receive the art so that I can authenticate it.
    4.2. As an authenticator I want to be able to update the status of an art piece to be verified so that it can be shipped.
5. Shipping
    5.1. As shipping I want to be able to collect the art off to the seller so that I can get it to the authenticator.
    5.2. As shipping I want to be able to collect the authenticated art off the authenticator so that I can get it to the buyer.
6. Payment handler
    6.1. As payment I want to be able to hold the buyers money until the art is verified so that I know they have the funds.
    6.2. As payment I want to be able to hold an insurance fee from the seller's money until the art is verified so that I can later keep it if it's fake.
7. System
    7.1. As system I want to be able to display art on users' feeds based on their preferences so that they can see what they like.
    7.2. As system I want to be able to rank sellers so that I can offer top performing sellers incentives.
    7.3. As system I want to be able to rank buyers so that I can offer top performing buyers incentives.
    7.4. As system I want to be able to make suggestions to sellers on their auction properties so that I can help them do well.

We have elected use cases 1.2, 2.6, 1.3, 3.2, 1.1, 2.3, 4.2, and 2.7 to elaborate on further as we believe they're the most important to our system.

Use case 1.2	**Make a sale**	
Goal in Context	Seller accepts a bidders bid, expects shipping to collect the art, have it authenticate and then receive the payment from the buyer.	
Scope & Level	Auctionly website, summary.	
Preconditions	We have seller banking information and address, we have buyers banking information and address.	
Success End Conditions	Authenticator authenticates the art and sale goes through.	
Failed End Conditions	Authenticator doesn't authenticate the art and sale is stopped.	
Primary, Secondary actors	Primary: Seller, Buyer. Secondary: Authenticator, Shipping, Payment handler.	
Trigger	Seller accepts a bidders bid.	
Description	**Step**	**Action**
	1	Seller uploads an art piece.
	2	Seller starts an auction on the art piece.
	3	Buyers begin bidding on art piece.
	4	Seller accepts a bid.
	5	Money is held from buyers account by payment handler.
	6	Insurance money is held from sellers account by payment handler.
	7	Shipping collects art piece from seller and delivers to authenticator.
	8	Authenticator authenticates the art piece.
	9	Shipping collects art piece from authenticator and delivers to buyer.
	10	Insurance fee is transferred back to sellers account.
	11	Buyers money is transferred to sellers account.
Extensions	**Step**	**Branching Action**

	5.a	Buyer doesn't have enough money in their account for the amount they bid. 5.a.1 cancel purchase.
	6.a	Seller doesn't have enough money in their account to cover the insurance fee. 6.a.1 cancel purchase.
	8.a	Authenticator find art piece to be fraudulent.
	9.a	Shipping returns art piece to seller.
	10.a	Insurance fee is transferred to treasury.
	11.a	Buyers money is transferred back to buyer.
Variations	**Step**	**Action**
	1	Seller may cancel an auction without accepting a bid.

Use case 2.6	**Receive incentives**	
Goal in Context	Give top performing sellers and buyers incentives like reduced commission rates.	
Scope & Level	Auctionly website, Primary task.	
Preconditions	Seller must be in top ranked. Buyer must be in top ranked.	
Success End Conditions	User receives incentive.	
Failed End Conditions	User doesn't receive incentive.	
Primary, Secondary actors	Primary: Buyer, Seller. Secondary: system.	
Trigger	Seller/Buyer is included in top ranking.	
Description	**Step**	**Action**
	1	System ranks Sellers/Buyers as top performing on auctionly.
	2	System selects top 10 sellers and top 10 buyers to receive incentives/discounts.
	3	Seller/Buyer receives a notification they have been selected.
	4	At next sale/purchase their discount is applied by system.
Extensions	**Step**	**Action**

	3.a	Seller/Buyer isn't in top performing. 3.a.1 doesn't receive an incentive.
Variations	**Step**	**Action**
	4	Don't make a sale/purchase in the time frame incentive is valid.

Use case 1.3	**Update auction**	
Goal in Context	A seller can update their auction properties to help it do better.	
Scope & Level	Auctionly website, subfunction	
Preconditions	We have seller banking information and address, they have begun an auction on an art piece.	
Success End Conditions	Properties updated.	
Failed End Conditions	Properties not updated.	
Primary, Secondary actors	Primary: seller. Secondary: system.	
Trigger	Seller clicks update properties button on auctions page.	
Description	**Step**	**Action**
	1	Seller starts an auction on an art piece.
	2	Seller clicks on update auction properties.
	3	System displays auction properties form.
	4	System suggests properties to seller.
	5	Seller selects from the suggested properties.
	6	Seller clicks save changes.
Extensions	**Step**	**Action**
	5.a	Seller enters their own properties.
Variations	**Step**	**Action**
	6	Seller cancels changing the properties.

Use case 3.2 **Actor**	**Create an account** **Browser**
Actor action	**System response**
1. Clicks create an account button. 3. Fills in fields. 4. Clicks register button.	2. Display sign up form. 5. Validates form inputs. 6. Stores account information. 7. Logs in user.
Alternative route	
4.a. Clicks cancel button.	5.a find inputs in valid and alerts user 6.a Returns user to feed.

Use case 1.1 **Actor**	**Upload art** **Seller**
Actor action	**System response**
1. Clicks upload new art button. 3. Fills in information for uploading their art. 4. Selects upload for exhibition. 5. Clicks upload.	2. Displays art uploading form. 6. Validates inputs. 7. Adds art to sellers' exhibitions. 8. Returns user to their exhibitions.
Alternative route	
5.a Clicks cancel.	6.a Returns user to their exhibitions.

Use case 2.3 **Actor**	**Make a bid** **Buyer**
Actor action	**System response**
1. Clicks on a piece of art that's up for auction. 3. Clicks make a bid button. 5. Enters a bid.	2. Displays the auction page with its properties. 4. Displays bid form. 6. Alerts seller to buyers bid.
Alternative route	
3.a exits auction page. 5.a cancels making a bid	

Use case 4.2 **Actor**	**Update states of art authentication process** **Authenticator**
Actor action	**System response**

1. Authenticator logs on to authentication system. 3. Authenticator updates an art piece to authenticated.	2. System displays the art pieces waiting for states update. 4. System alerts shipping to collect art piece and ship to buyer. 5. System alerts payment handler to return insurance fee and transfer buyer's money.
Alternative route	
3.a Autehnticator updates art piece to not authentic.	4.a System alerts shipping to collect art piece and ship to seller. 5.a System alerts payment handler to return buyers money and transfer insurance fee to treasury.

Use case 2.7 **Actor**	**Specify preferences** **Buyer**
Actor action	**System response**
1. Buyer clicks on their profile. 3. Buyer clicks update art preferences. 5. Buyer adds preferences.	2. System displays their profile page. 4. System displays preferences form. 6. System registers the update.
Alternative route	
5.a Buyer clicks cancel.	

Functional Requirements

1. System shall allow users to view a feed of the available art that's up for auction and exhibition and order the feed accordingly.
   Allocated use cases: 1.4, 2.2, 3.1, 7.1
2. System shall allow browsers to create an account and register as a buyer or seller.
   Allocated use cases: 3.2
3. System shall allow buyers to specify their art preferences.
   Allocated use cases: 2.7, 1.7, 7.1
4. System shall allow users that are registered as sellers to upload art and set its properties.
   Allocated use cases: 1.1, 1.7
5. System shall allow sellers to begin an auction on their art and customise its properties throughout the auction process.
   Allocated use cases: 1.3, 1.8, 7.4
6. System shall allow the seller to accept a buyer's bid and sell their art.
   Allocated use cases: 1.2, 6.2, 2.3
7. System shall allow buyers to place bids on art for a chance to buy the art.
   Allocated use cases: 2.3, 1.2, 6.1, 1.8

8. System shall handle setting up shipping of the art pieces between the seller, authenticator and buyer.
   Allocated use cases: 1.6, 2.5, 4.1, 5.1, 5.2
9. System shall handle setting up authentication of the art pieces by an authenticator before completing the sale.
   Allocated use cases: 2.4, 4.1
10. System shall allow an authenticator to update the status of an art piece so that it can progress to shipping.
    Allocated use cases: 4.2, 5.2
11. System shall display the art bought by a buyer on their profile.
    Allocated use cases: 2.1
12. System shall monitor sellers and buyers progress and rank them accordingly.
    Allocated use cases: 7.2, 7.3
13. System shall offer incentives to top ranking sellers and buyers.
    Allocated use cases: 7.2, 7.3, 2.6, 1.5

Non-functional

1. System shall be secure: the system will be secure from misuse and unauthorised access.
2. System shall be extensible: the system's internal structure and dataflow will not be affected by new or modified functionality.
3. System shall be reliable: the system will satisfactorily perform the functional requirements for which it was designed and intended.
4. System shall be maintainable: the system's software will be clear and understandable with minimal effort to ensure it can be easily modified.

Tactics to support quality attributes

1. *Security*

   To ensure the security of our system we will put in place procedures to protect us against the OWASP(Open Web Application Security Project) top 10 information security risks.
   Therefore we'll protect against:
   1) Injection.
      We will validate all user data that is manually inputted through forms on our website to ensure there is no malicious query embedded in the input.
   2) Broken Authentication.
      To ensure some can't get access to a user's account through a stolen user identifier, we won't allow more than one session to be active for each user.
   3) Sensitive Data Exposure.

We will encrypt user's sensitive information such as passwords and banking information to ensure against hackers gaining access to the information while it's being transmitted over HTTP protocol.

4) XML External Entities.
   To ensure against a XXE attack, we won't be using any xml files in our production of our system.
5) Broken Access Control.
   To ensure against attackers gaining access to privileged pages on our website we will have a list of IP addresses and device IDs of those allowed access.
6) Security Misconfiguration.
   As default settings in web servers are often insecure, we will ensure we enable and disable appropriate settings while developing our system. For example, we will have cookies http only enabled to defend against attackers accessing and stealing user session cookie through javascript.
7) Cross-Site scripting.
   To protect against XSS we will implement a strong Content Security Policy located in the HTTP header to provide instructions for compatible browsers on which domains and commands to trust.
8) Insecure Deserialization.
   To protect against deserialization attacks we will limit the classes that are allowed to deserialize to those that it's necessary for their functionality.
9) Using Components with known Vulnerabilities.
   We will be using the latest versions of the external components used in our system and be sure to keep up to date with any vulnerabilities found.
10) Insufficient Logging and Monitoring.
    We will regularly test our logging by having a pentester breach our system and ensure we can view that in our logs.

2. *Extensibility*

To ensure the extensibility of our system we will:

1) Use modules to separate the functionality of our system into independent, interchangeable and highly cohesive modules.
2) The system architecture is loosely coupled, components are weakly associated with each other.
3) Use Python as it is modern, easy to use, and object oriented to assist in the modularity of the system.

3. *Reliability*

To ensure the reliability of our system we will:

1) Test the system regularly throughout the development process to ensure each functional requirement is met.
2) Use modularity to aid in the testing of the system.

3) Take a dev ops approach to the development of the system to provide communication, close operation, integration and automation among all developers to plan, develop, test, deploy, release, and maintain the system.

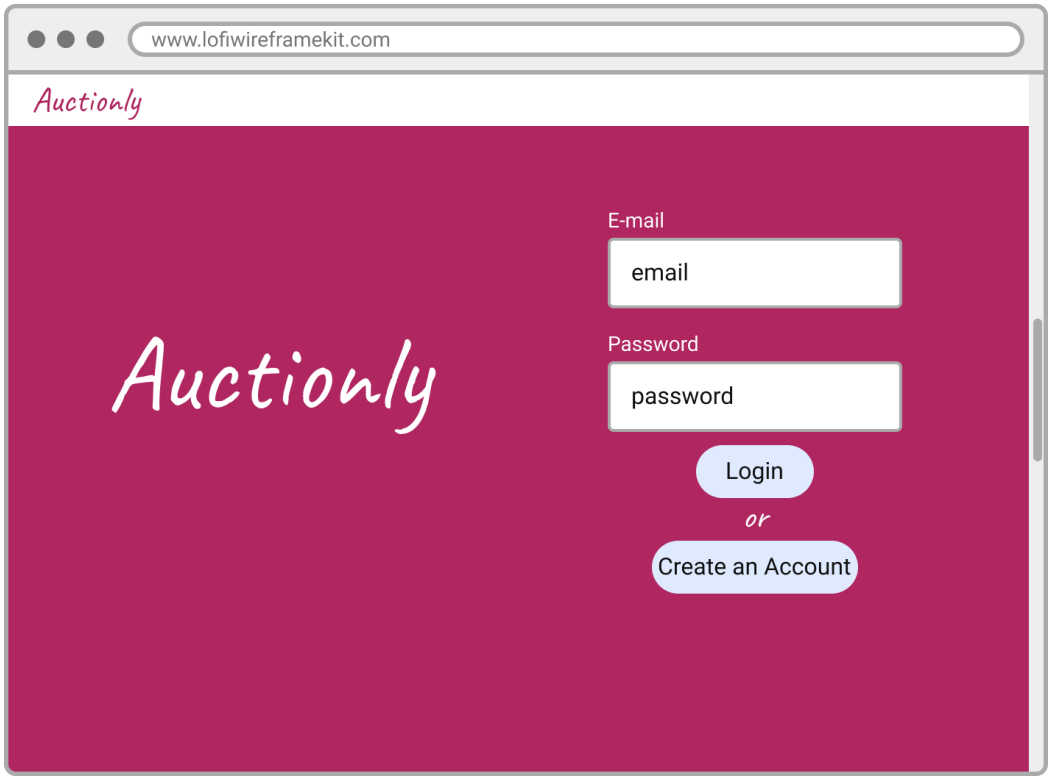
4. *Maintainability*

To ensure the systems maintainability we will:

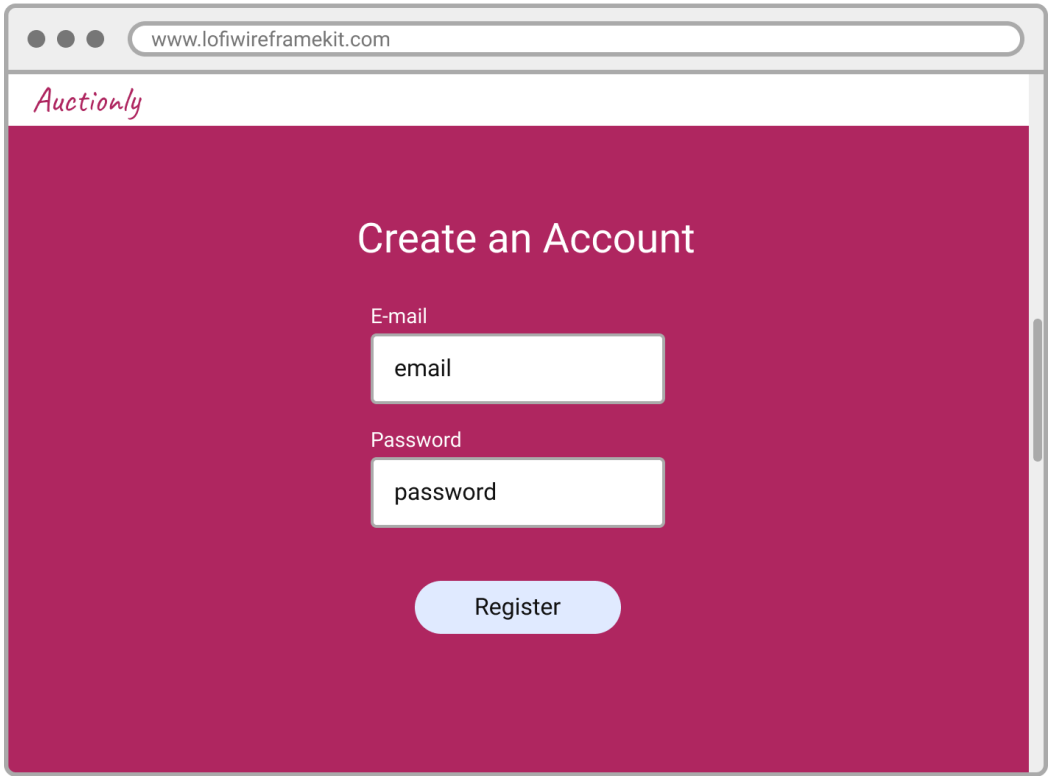
1) Use modularity to ensure each component is focused on just its functionality and easy to understand.
2) Make use of polymorphism to help ensure changes to server side implementation/business logic doesn't require changes to clients side also.
3) Use interfaces where applicable.
4) Use comments throughout our code to ensure it's clear what the code is doing and the thought process behind implementing it.
5) Use interfaces to define the features of the modules on which clients can rely and control dependency management.
6) Keep the code base as small as possible.
7) Automate tests for the system.

Sample GUI prototypes

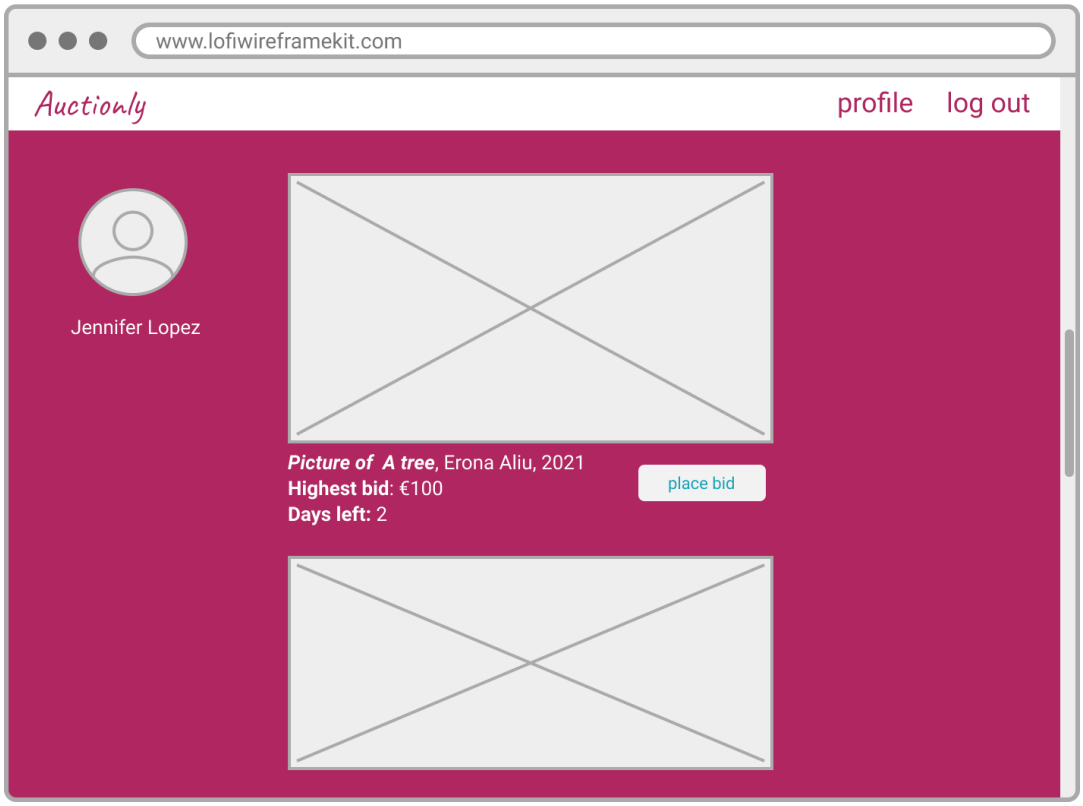
Landing Page



Register Page



Home Feed Page



Discussion of System Architecture

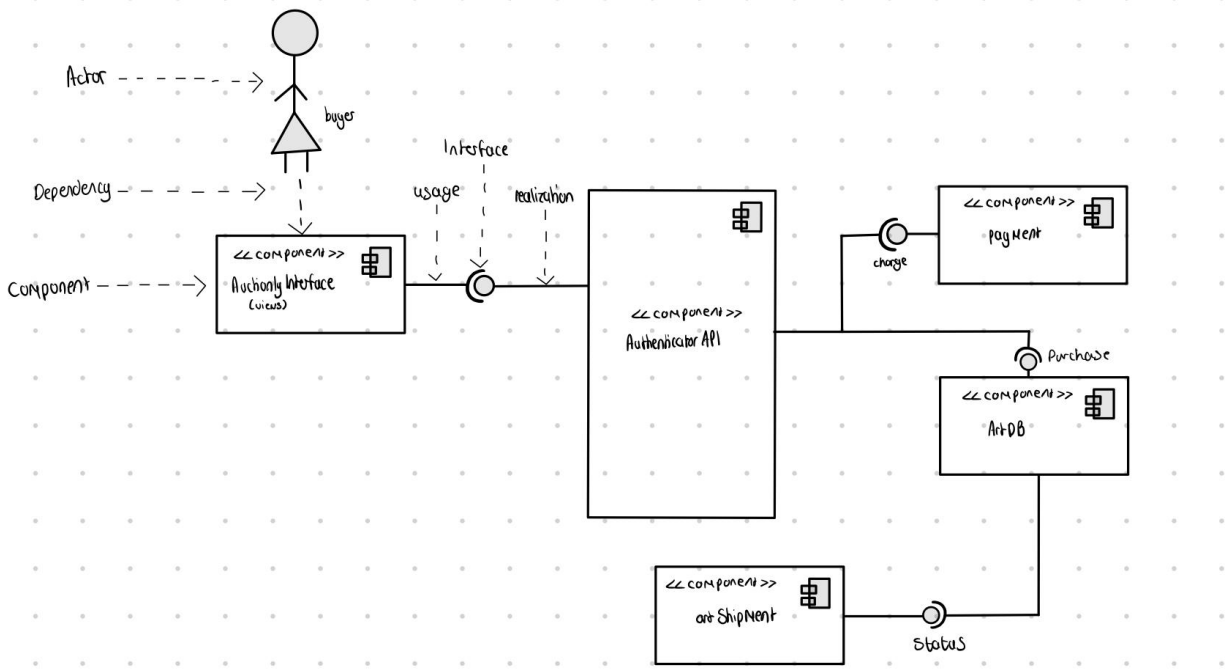
The system architecture is the structural design of our system in which systems are a class of software that provide the foundational services as well as automation. Below, we have some illustrative examples of the Auctionly system architecture.

Components

Our components have been designed to be reusable and they also allow us to divide and conquer large problems into smaller ones thus easing our development process. Below is an example of how a user makes a bid from the UI and what happens in more detailed circumstances.

Components can be wired together to form subsystems with the use of the ball and socket joint.[1]
Provided interfaces: Interfaces where a component produces information used by the required interface of another component are represented by a straight line with an attached circle.
Required interfaces: A straight line from the component box with an arc at the end represents the interfaces where a component requires information in order to perform its proper function.

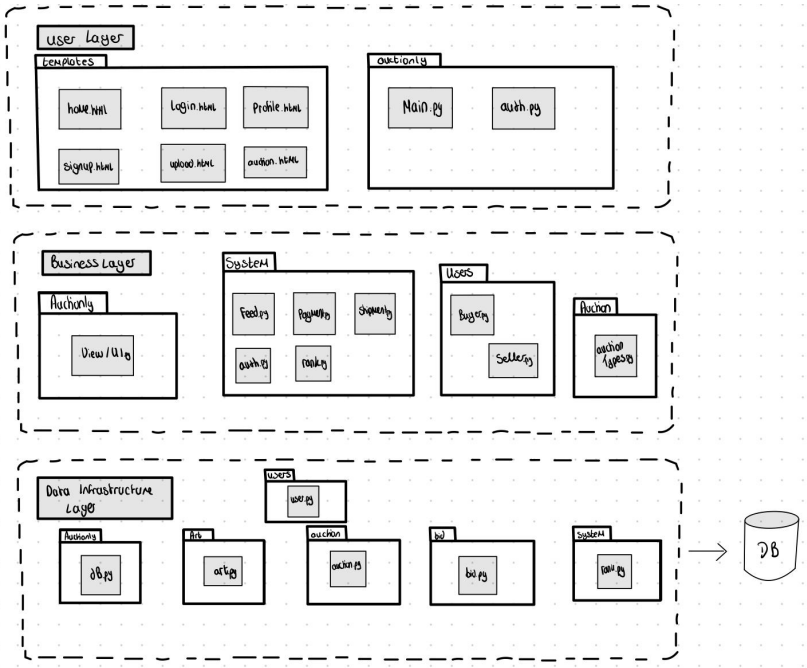
Below, we have an authenticator API component which authenticates a piece of art from a buyer's winning bid. A component that processes the credit card charges and the database that contains the art information sets the art in question to be authenticated and bought, then our art is shipped to the buyer's address.[2]



Layers

We used the three tier architecture which is a client-server architecture where Auctionly is separated into three logical and physical computing tiers:

1. **User:** user interface and communication layer where the end user interacts with Auctionly
2. **Business:** where the data is processed
3. **Data Infrastructure:** where data associated with the project is managed and stored

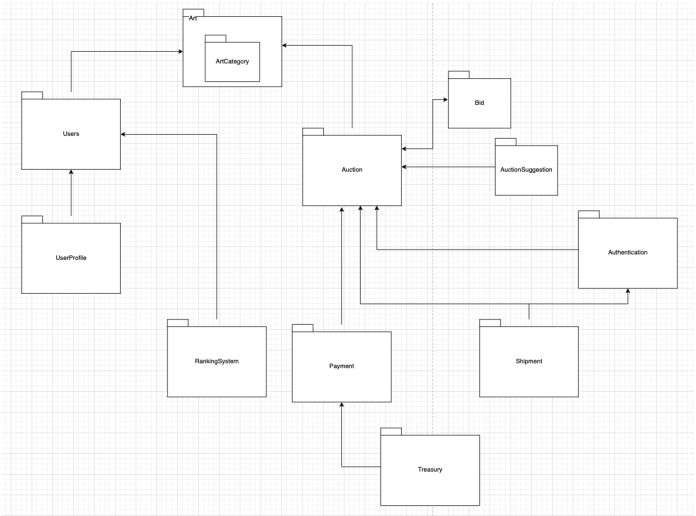


Services

Auctionly provides many services to users. It provides users with auction suggestions, bidding options, selling art, authentication on art pieces, payment options, view art available for auction on a feed, rewards etc. These services are loosely coupled so we can edit and rework a service without affecting others.

Package Diagram

A diagram of the packages we expect to have in our system and what packages they each work with.



Analysis sketches

Candidate Class Identification using Data Driven Design

In this section, we used the noun identification technique *(class names: nouns, functions: verbs)* to recognize and note down nouns present in the functional requirements. The discovered nouns were taken into account and discussions arose around which were most relevant to configure the classes.

We allow artists to exhibit their work through our website, which then reaches our customer base. Artists have the ability to create a profile on our website, and describe personal details about themselves and their art to allow buyers to get to know them. They have the power to publish their art, fully customise their auction properties - or pick one of the auction types designed by us. We rank our sellers by popularity and offer commission discounts to our most desirable artists. We also keep close track of the auction progress and provide suggestions if an art piece is not performing well during the auction.

Buyers are also able to create personal accounts on our website, where they can specify the type of art they are interested in. They are allowed to post bids on auctions and purchase art pieces when they win. We encourage our buyers by displaying a collection of their art pieces under their profile. We also rank our customers by activity and offer special discounts to our most loyal and determined auctioneers.

Nouns: **art**, **art pieces**, **artists**, **work**, **customer base**, **profile**, **personal details**, **auction properties**, **auction types, sellers, commission**, **auction progress**, **suggestions**, **auction**, **personal accounts**, **buyers**, **type of art**, **bids, art pieces**, c**ollection of art pieces, profile, activity**, **discounts**.

Class List Discussion Output

Class A	Class C **[** Quite Vague **]**
Class B **[** Might be irrelevant **]**	Class E **[** Too Verbose **]**

→ Irrelevant

→ Up for consideration

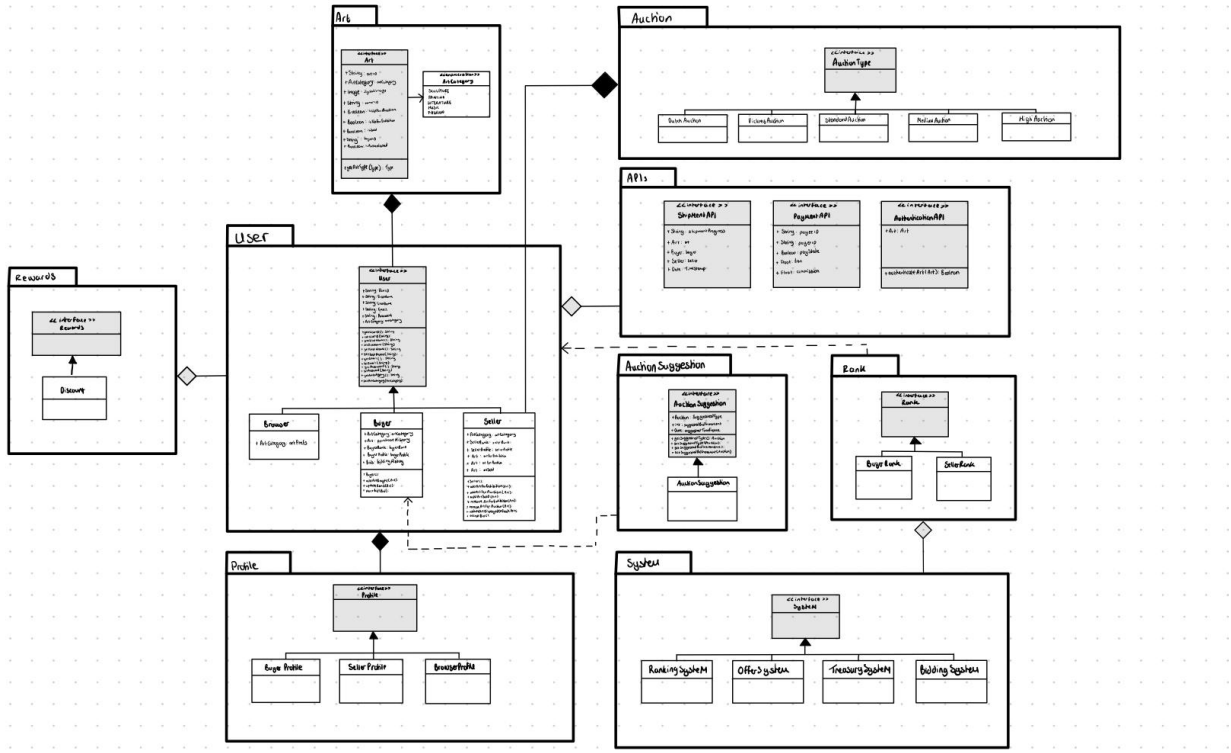
→ Implement - the ones we keep

[…] → Further Description on the class

Final Configuration of the Class List

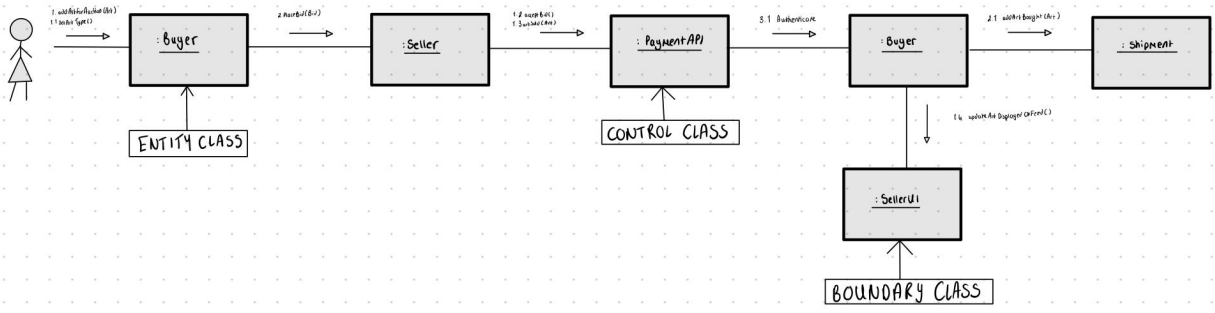
User [Need users, very relevant]	Seller [Seller to sell art, very relevant]	Browser [Account to browse art, might be irrelevant]
Buyer [Buyer with ability to bid and buy art, very relevant]	Profile [Display different user profiles, very relevant]	BuyerProfile [ability to bid on art, very relevant]
BrowserProfile [feed only, might be irrelevant]	SellerProfile [ability to upload art, very relevant]	DutchAuction [dutch auction method, very relevant]
AuctionSuggestions [suggestions for buyers, very relevant]	AuctionType [specific auction types, very relevant]	MediumAuction [medium auction method, might be too verbose]
VickreyAuction [vickrey auction method, might be too verbose]	StandardAuction [standard auction method, very relevant]	BuyerRank [ranking for buyer, very relevant]
HighAuction [high auction, might be too verbose]	Rank [ranking system, very relevant]	ArtCategory [art categories, very relevant]
SellerRank [ranking for seller, very relevant]	Art [art class, very relevant]	Bid [bidding ability for buyers, very relevant]
Rewards [rewards for users, very relevant]	Discount [discount reward for users, very relevant]	RankingSystem [ranking system, very relevant]
System [system interface, very relevant]	TreasurySystem [treasury system, very relevant]	ShipmentAPI [shipment api for sellers and buyers, very relevant]
OfferSystem [offer system, too verbose]	PaymentAP [payment api for buyers, very relevant]	AuthenticationAPI [authenticate art for buyers, very relevant]

Analysis Class Diagram (Prior to Implementation)



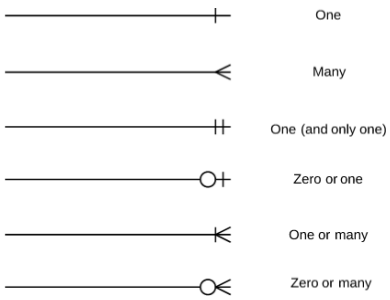
Communication Diagram

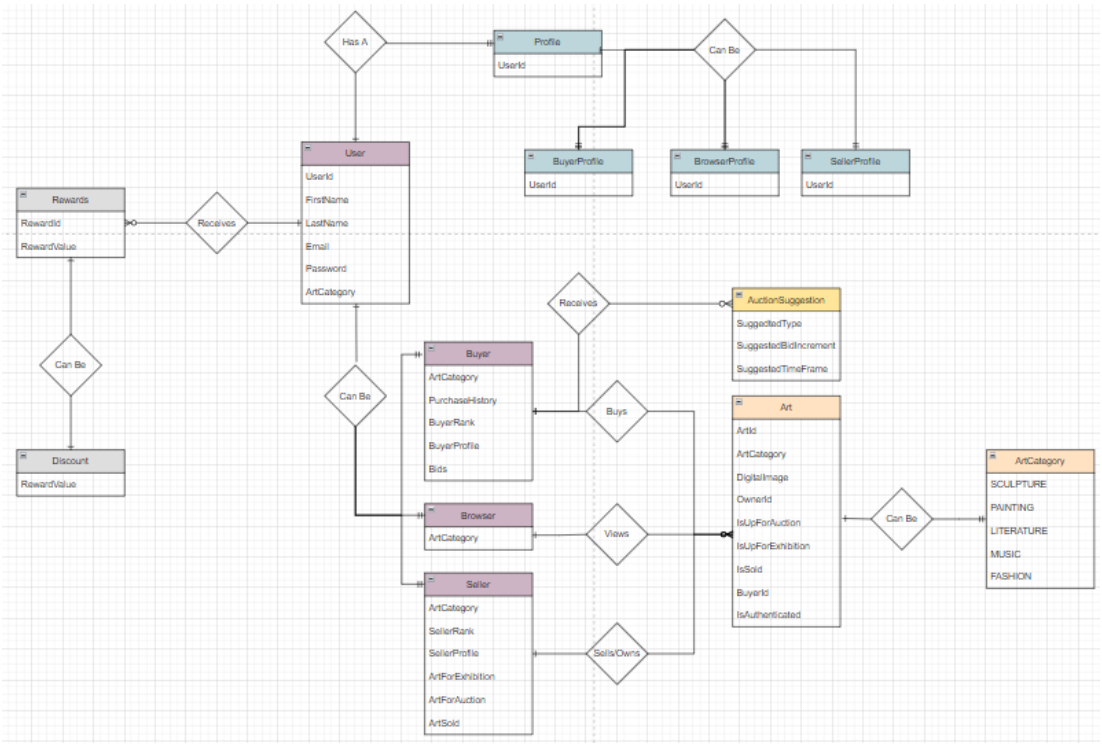
A communication diagram, like a sequence diagram, shows us the interaction between objects. Below, we have a sample from the seller. We have a seller who uploads art and then puts it up for auction by setting its auction type. A potential buyer will then see the auction and place a bid. Once the bid is placed, the buyer with the highest bid after an allocated time will be selected. The PaymentAPI will automatically receive the funds from the buyer's predefined details once the user is authenticated. The art piece will then be marked as sold, the feed will be updated and the ShipmentAPI will be called in order to ship the painting to the buyer's predefined address.



Entity Relationship Diagram with Cardinality

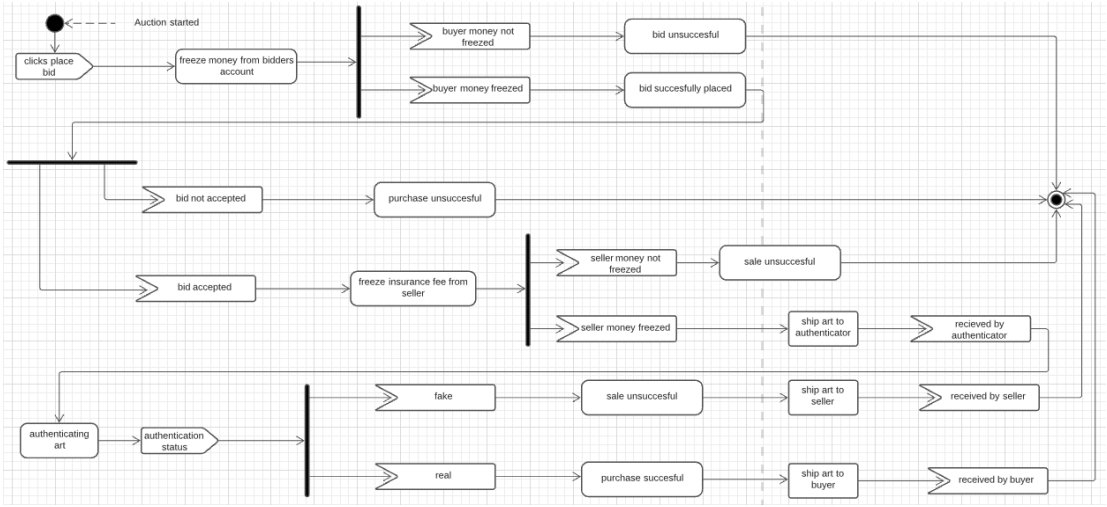
Cardinality defines the possible number of occurrences in one entity which is associated with the number of occurrences in another. (One user has many paintings, One artist has many paintings) → ER diagram, cardinality represented as a crow's foot at the connector's end.





Activity Diagram

An activity diagram of the process of purchasing/selling an art piece.



List of Classes

A listing in tabular format of classes in each package, their authors, and lines of code. Also include total lines of code developed (a) for the application as a whole, and (b) per team member.

Package	Class	Authors	Lines of Code
Users	User	Nutsa	97
	Seller	Erona	73
	Buyer	Erona	49
	UserPreference	Ranya	23
Art	Art	Ranya	56
	ArtNotifications	Ranya	23
Auction	Auction	Nutsa, John	255 (127.5)
	DutchAuction	Nutsa	27
	AuctionState	John	29
Bid	Bid	Nutsa	38
	BidCollection	Nutsa	60
System	Authentication	Nutsa	23
	Feed	Ranya	44
	Payment	Nutsa, John	119 (59.5)
	Rank	Ranya	151
	RankedUser	Ranya	36
	Sale	John	25
	Shipment	Nutsa	25
Templates	auction-art.html	Erona	31
	auction.html	Nutsa, John	117 (58.5)
	base.html	John	54
	edit-auction.html	Nutsa, John	24 (12)

	home.html	Ranya	74
	login.html	John	15
	profile.html	Erona	115
	signup.html	John	53
	upload.html	Erona	31
Auth	Auth	John	116
Database	database.db	All Members	-
View	Views	All Members	229 (57.25)
Main	Main	John	8
Workflows	pylint.yml	All Members	25 (6.25)
	unit_testing.yml	All Members	27 (6.75)
Tests	Test_Users	All Members	45 (11.25)
	Test_Payment	All Members	16 (4)
Fee Constants	FeeConstants	Nutsa	9
Lines of code developed by Ranya			488.5
Lines of code developed by Nutsa			632.5
Lines of code developed by Erona			380.5
Lines of code developed by John			624.5
Total lines of code developed			2,126

Disclaimer:

A lot of our work we did together and spent numerous hours on Microsoft Teams calls helping each other with different aspects of the code. We truly feel equal effort was put in by all team members irrespective of lines of code and commits. Also depending on roles assigned some were more coding related than others.

Code

Design patterns

State Design Pattern

The state design pattern is a pattern that allows an object to change its functionality depending on the state it is currently in. The state is implemented as a derived class of the state interface. If there is a change in state we can override the interface to execute different logic under the same method. The state of our auctions has the ability to be updated more easily, and allows for future iterations of our code to include methods where the functionality will change depending on the state. The state design pattern also allows other classes to consume the state confidently, knowing every possible state the auction can be in to perform different actions based on the state.

```python
from abc import ABC, abstractmethod


class IAuctionState(ABC):
    @staticmethod
    @abstractmethod
    def __str__():
        pass

class Started(IAuctionState):
    def __str__(self):
        return "Started"

class Ended(IAuctionState):
    def __str__(self):
        return "Ended"

class Shipped(IAuctionState):
    def __str__(self):
        return "Shipped"

class Authenticated(IAuctionState):
    def __str__(self):
        return "Authenticated"

class Rejected(IAuctionState):
    def __str__(self):
        return "Rejected"
```

Observer design pattern

Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically.[4]
When learning about observer we knew it would fit in perfectly with our project as we wanted to alert users when an art piece they're interested in goes on auction.

Because we were implementing a website we had to get a bit creative with how we implemented the pattern through html and storing it in a database.

Here we have our subject which is an art piece:

art > art.py > Art > get_image

```python
"""module containing art class"""
from auctionly import db # pylint: disable=E0401

class Art(db.Model):
    """Art class implemented to create art objects that can be passed between users"""
    id = db.Column(db.Integer, primary_key=True)
    owner_id = db.Column(db.Integer, db.ForeignKey("user.id"))
    name = db.Column(db.String(150))
    description = db.Column(db.String(150))
    image = db.Column(db.String(150))
    art_category = db.Column(db.String(150))
    up_for_auction = db.Column(db.String(5))

    def __init__(self, name, owner_id, digital_image_path, description, art_category):
        """creates an art object"""
        self.name = name
        self.owner_id = owner_id
        self.image = digital_image_path
        self.description = description
        self.up_for_auction = "False"
        self.art_category = art_category
        self.art_status = "With owner"
```

This is how we created an art piece and then stored it in our database.

We then created a table to store the list of observers (users) for an art piece:

art > art_notifications.py > ...

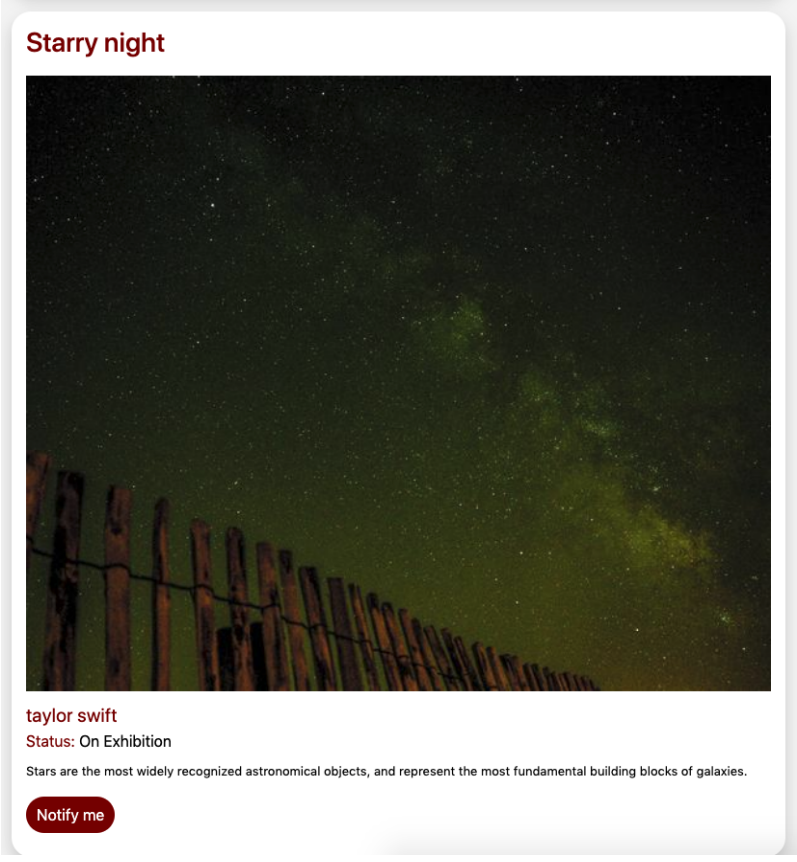
```python
"""Art notification module implemented to hold Art Notifications class"""
from auctionly import db # pylint: disable=E0401

class ArtNotifications(db.Model):
    """Art notification class implemented to create art notification"""
    # pylint: disable=E1101
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey("user.id"))
    art_id = db.Column(db.Integer, db.ForeignKey("art.id"))

    def __init__(self, user_id, art_id):
        """creates an art notification object/row in the table"""
        self.user_id = user_id
        self.art_id = art_id
```

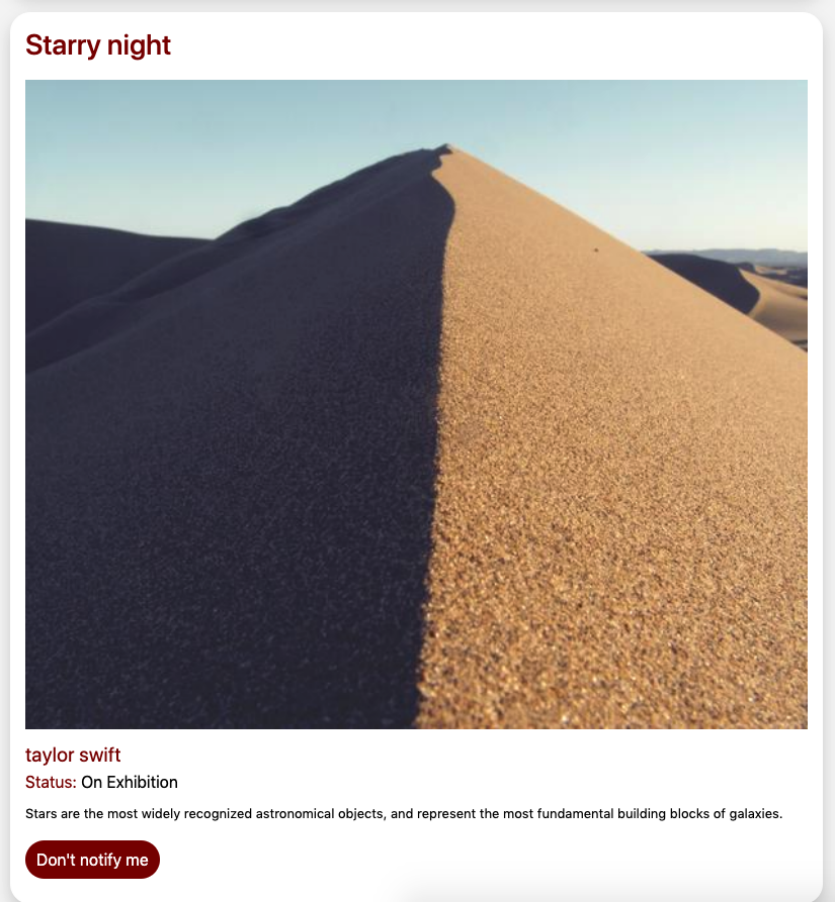
This kept track of who wanted to be notified of what and also acted as a change manager between the observers and the subject.

Here we have our observers:

```
class User(db.Model, UserMixin):
    """ initializing the user class and providing getters and setters to
    interact with the users model in the database.  """
    __tablename__ = 'user'

    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(150), unique=True)
    password = db.Column(db.String(150))
    first_name = db.Column(db.String(150))
    last_name = db.Column(db.String(150))
    user_type = db.Column(db.String)

    def __init__(self, first_name, last_name, email, password):
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.password = password
        self.feed = []
```

Our observers were the users themselves and they could attach and detach themselves from subjects (art notifications) by clicking on a “notify me” button to attach and a “Don’t notify me button” to detach.





(Please excuse the different images as we used a random image generator for our prototype therefore the images change each time an update happens.)

Here is how we handled attach and detach requests:

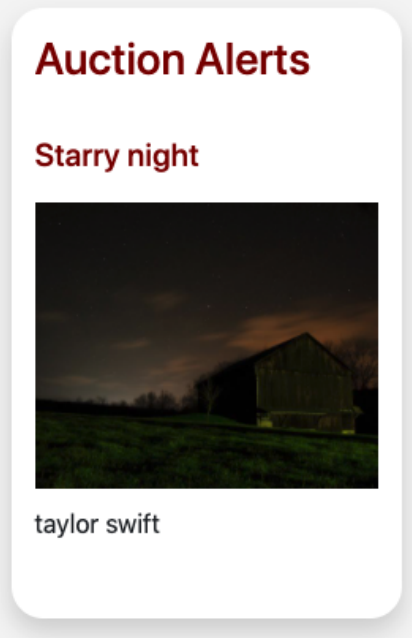
```
41          # checking is the reuquest is to be added to the arts observer list
42          # and handling it
43          if(notify == "True"):
44              attach = ArtNotifications(flask_login.current_user.id, art_id)
45              db.session.add(attach)
46              db.session.commit()
47              user_notifications = user.get_notification_list()
48              message = "You have been added to the notifications list for " + \
49                  Art.query.filter_by(id=art_id).first().get_name() + "."
50              flash(message, category="success")
51
52          # checking is the reuquest is to be removed from the arts observer list
53          # and handling it
54          elif(notify == "False"):
55              ArtNotifications.query.filter((ArtNotifications.art_id == art_id) & (
56                  ArtNotifications.user_id == flask_login.current_user.id)).delete()
57              user_notifications = user.get_notification_list()
58              message = "You have been removed from the notifications list for " + \
59                  Art.query.filter_by(id=art_id).first().get_name() + "."
60              flash(message, category="error")
```

Once a change happens we notify the user and alert them:

```
72      def get_auction_notification_list(self):
73          """ returns the notifications of the user """
74          user_subs = ArtNotifications.query.filter_by(user_id=self.id).all()
75          user_notifications = []
76          for sub in user_subs:
77              art_id = sub.get_art_id()
78              print("art_id")
79              print(art_id)
80              art = Art.query.filter_by(id=art_id).first()
81              if art.get_up_for_auction() == "True":
82                  user_notifications.append(art)
83          return user_notifications
```

Here we got the list of all the art that has changed state and is now on auction.

Finally, we display it on the UI as follows.



Iterator Design Pattern

The iterator design pattern is one of the most popular design patterns due to its simplicity and efficiency. It allows the system to iterate through a collection of objects without exposing it's internal details. We opted in for using this pattern for our Bid class. The iterator class works with Bid objects and provides a way to externally iterate over a collection of Bids and return the bid id of Bid objects that have been placed on a concrete auction. This can have many uses, such as displaying the number of bids that have been placed on the auction or revealing the details of the ongoing auction process to the seller.

```python
class BidIterator(Iterator):
    """ Concrete iterator to iterate over the Bid objects """
    _position: int = None
    _reverse: bool = False

    def __init__(self, collection: BidCollection, reverse: bool = False)
        self._collection = collection
        self._reverse = reverse
        self._position = -1 if reverse else 0

    def __next__(self):
        """ return next iterable object """
        try:
            value = self._collection[self._position].get_bid_id()
            if self._reverse:
                self._position += -1
            else:
                self._position += 1
        except IndexError:
            raise StopIteration()

        return value

    def has_next(self):
        """ check if iterator has an object left to iterate over """
        if self._reverse:
            return self._index >= 0
        return self._index < len(self._collection)
```

fig: definition of the iterator class

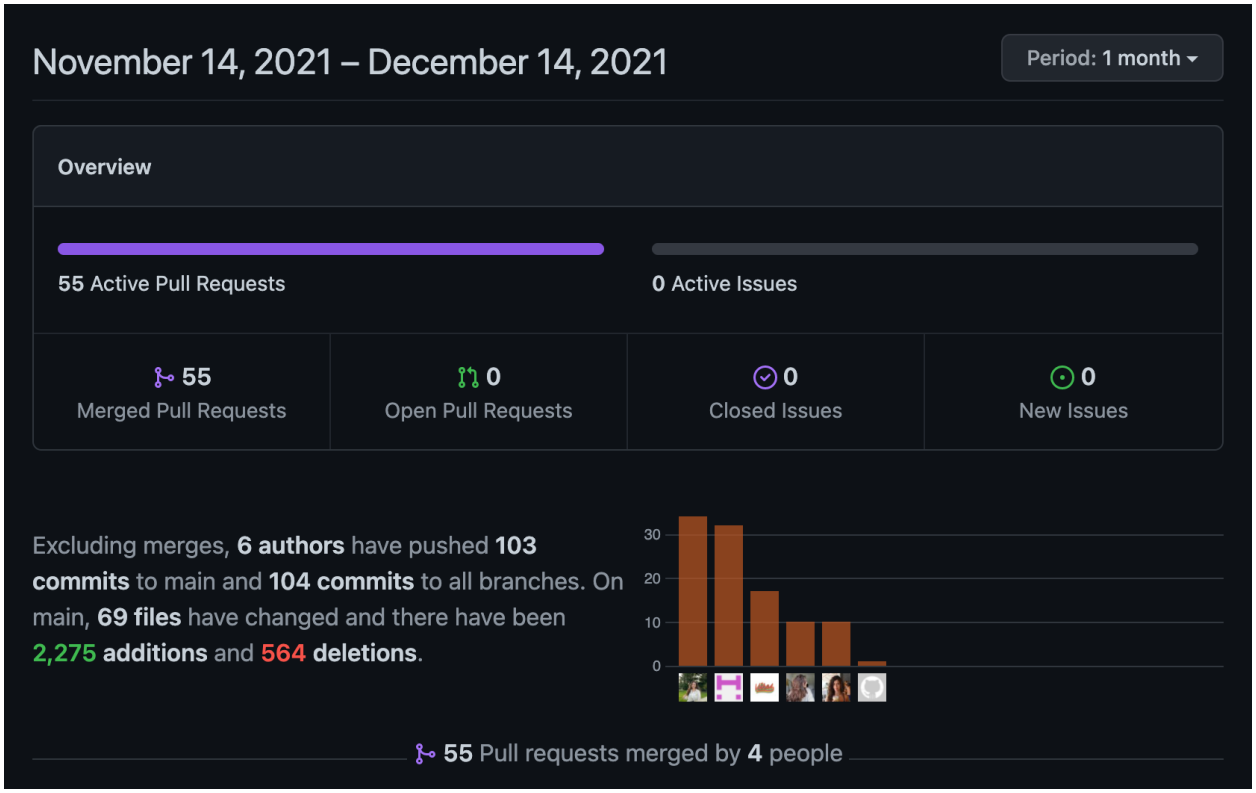
```python
def iterate_over_bids(self):
    """ method to print id's of all the bids that have been placed on the auction """
    collection = self.get_bids()
    iterator = BidIterator(collection=collection, reverse=False)
    while iterator.has_next():
        print(iterator.__next__())
    iterator.__next__()
```

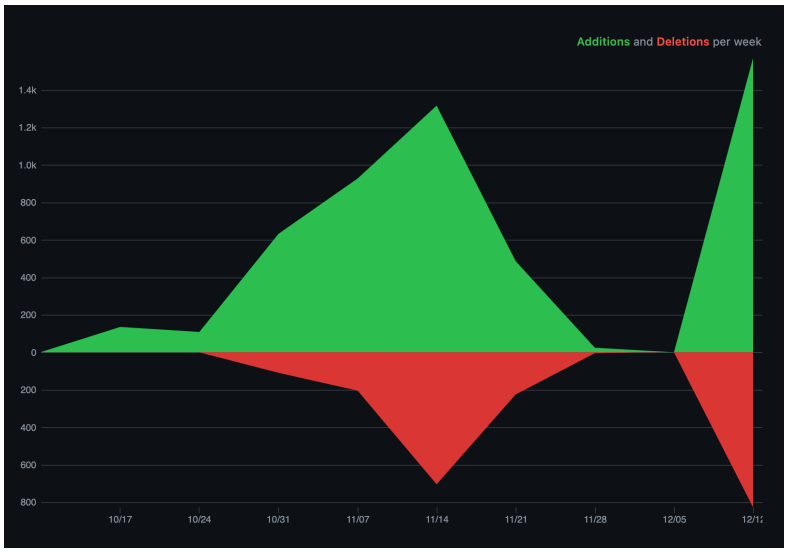
fig: external use of the iterator

GitHub (Project Hosting)

GitHub has been a tremendous asset for us during the development of our project. It allowed us to stay up to date and connected despite the fact that we were working remotely, in separate places. We ensured to commit and merge any code changes, as well as keep an eye on other teammates' changes to ensure transparency.

The figure below shows the insights for our Auctionly GitHub repository for the past month alone.





You can see from the chart that we heavily benefited from refactoring. We jumped straight into coding as soon as we could, and changed/deleted the lines of code that we deemed unnecessary or wrong as the development stage progressed. This saved us time on unnecessary detailed planning sessions and commitments.

It is important to note that we had issues with GitHub recognizing our commits at the beginning stages of development, therefore we don't feel that these numbers fairly describe the shares of work we put into developing the project and unanimously agree that our work was fairly distributed amongst all four of us.

Model View Controller (MVC)

We use the Model View Controller architectural pattern to support the interactivity of our system. This pattern helps us to seamlessly transfer information and data throughout the different layers of our system. We built the backend of the system before beginning to work on its incorporation with a front-end system, therefore our core functionality is intact from the sample GUI we added to our system.

We display a lot of information to the user with the help of the MVC pattern, the users are able to view each auction, their profile, their personalized feed, which seamlessly interact with and fetch information from the back-end of the system.

You can see below an example of how we were able to connect the user-facing and developer-facing components of the system. With the help of HTML, Jinja and Flask we were able to call python methods inside the existing HTML files.

```
<br><br> {% if auction.able_to_place_bid(user): %}
<form method="POST">
    <input type="submit" name="auction_action" value="Place €{{auction.get_current_bidding_price()}}
</form>
{% elif auction.has_user_won_auction(user): %}
<h8 class="mb-2" style="bold">This auction has ended. Claim art below. </h8>
<form method="POST">
    <input type="submit" name='claim_art' value="Claim Art" />
</form>

{% elif auction.has_timed_out() and auction.is_own_auction(user): %} {% if not auction.payment_has_be
<h8 class="mb-2" style="bold">Your auction has ended. Claim payment below. </h8>
<form method="POST">
    <input type="submit" name='claim_payment' value="Claim Payment" />
</form>

{% else %}
<h8 class="mb-2" style="bold">Your auction has ended. You have already claimed rewards. </h8>
{% endif %} {% elif auction.is_own_auction(user): %}
<form method="POST">
    <input type="submit" name="auction_action" value="Edit Auction" />
</form>
{% endif %}
```

And inside our Views.py, we were able to render and reroute html pages to support the user navigating throughout the system by clicking through the pages.

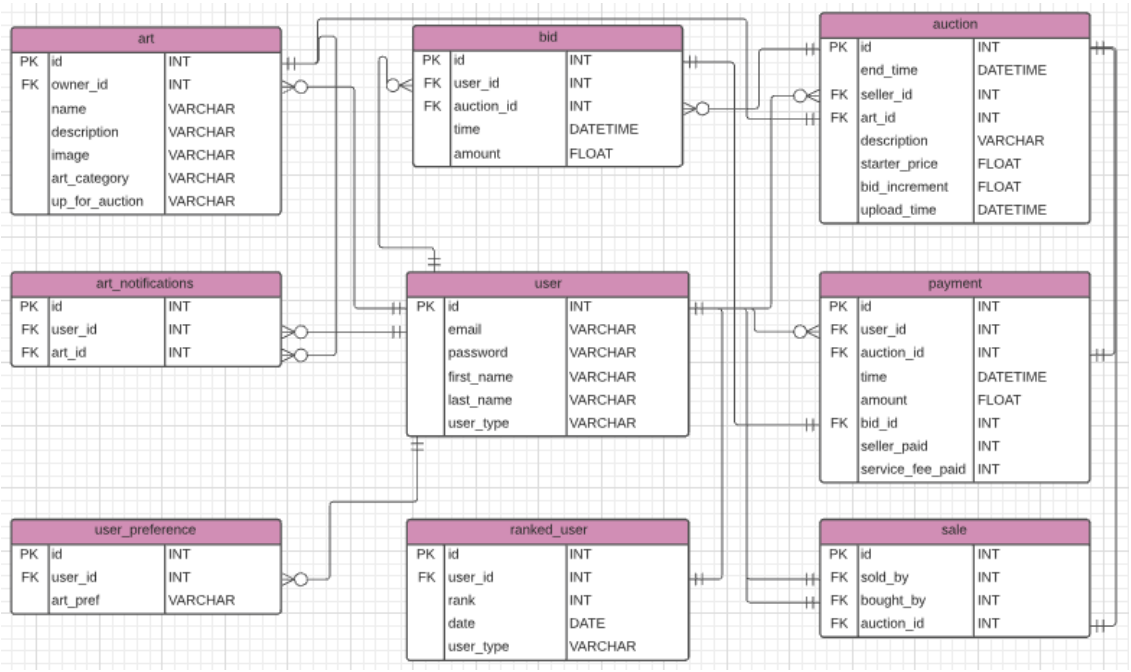
```python
@login_required
def home():
    """Handle home page of the website."""
    # extracting logged in user information
    user = flask_login.current_user
    user_pref = user.get_user_prefs()

    # prepping the users feed
    feed = Feed(flask_login.current_user.id)
    if len(user_pref) == 0:
        user_feed = feed.get_feed()
    else:
        user_feed = feed.get_users_feed(user_pref)

    # prepping the users notifications
    user_notifications = user.get_notification_list()
    user_auction_alerts = user.get_auction_notification_list()

    # checking if a notify request had been made on an art piece
    if request.args.get("art_id") is not None:
        art_id = request.args.get('art_id')
        notify = request.args.get('notify')

        # checking is the reuquest is to be added to the arts observer list
        # and handling it
        if notify == "True":
            attach = ArtNotifications(flask_login.current_user.id, art_id)
            db.session.add(attach)
            db.session.commit()
            user_notifications = user.get_notification_list()
            message = "You have been added to the notifications list for " + \
                Art.query.filter_by(id=art_id).first().get_name() + "."
            flash(message, category="success")
```

```python
@views.route('/auction-art', methods=['GET', 'POST'])
@login_required
def auction_art():
    """ upload auction page """
    if request.method == 'POST':
        art_id = request.form.get('artId')
        starting_price = request.form.get('startingPrice')
        bid_increment = request.form.get('bidIncrement')
        end_time = datetime.datetime.strptime(
            str(request.form.get('endTime')), "%Y-%m-%dT%H:%M")
        description = request.form.get('description')
        seller_id = flask_login.current_user.id

        new_auction = Auction(end_time, seller_id, art_id,
                              description, starting_price, bid_increment)

        db.session.add(new_auction)
        db.session.commit()

        # updating that this art piece is now on auction
        # iniates notifying the observers
        art = Art.query.filter_by(id=art_id).first()
        art.up_for_auction = "True"
        db.session.commit()

    user = flask_login.current_user
    user_art = user.get_user_art()
    for art in user_art:
        print(art.get_description())

    return render_template("auction-art.html", user=user, user_art=user_art)
```

Interesting aspects of implementation

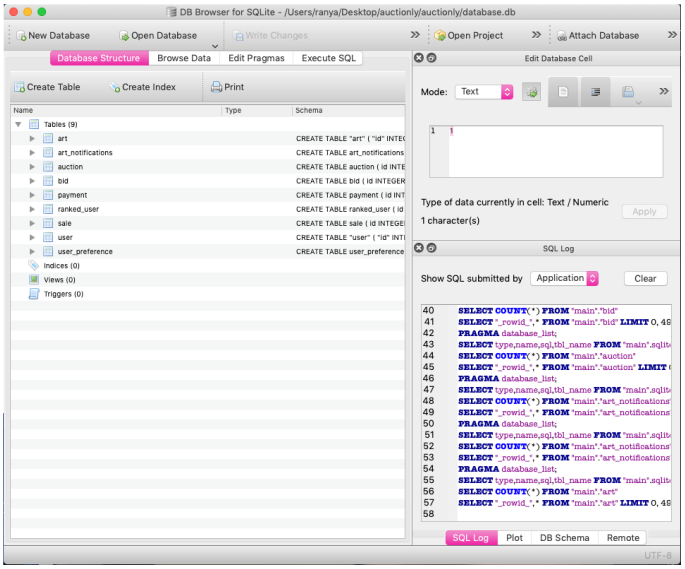
Database

We think an interesting aspect of our implementation is our fully functional database. We spent time on mocking up database tables to store our data and made sure it was normalised.

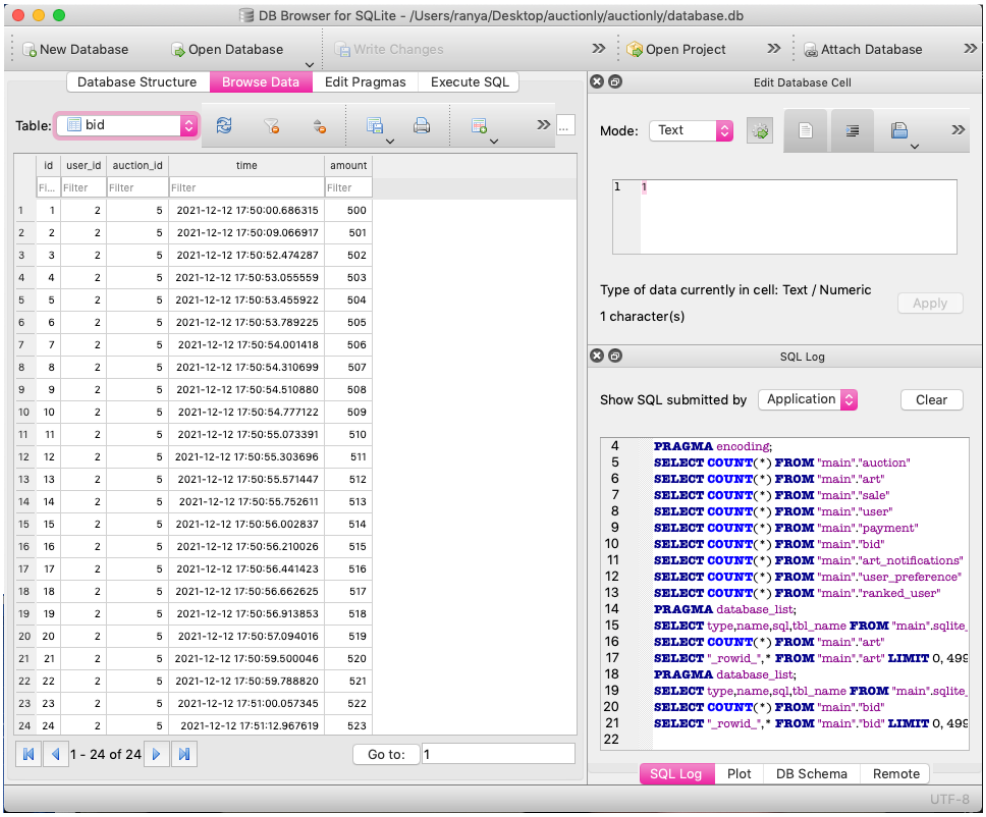


The database we used was SqlAlchemy and we were able to store objects as we created them in the database and push our new data to GitHub. We used the DB browser for SQLite to browse our database.

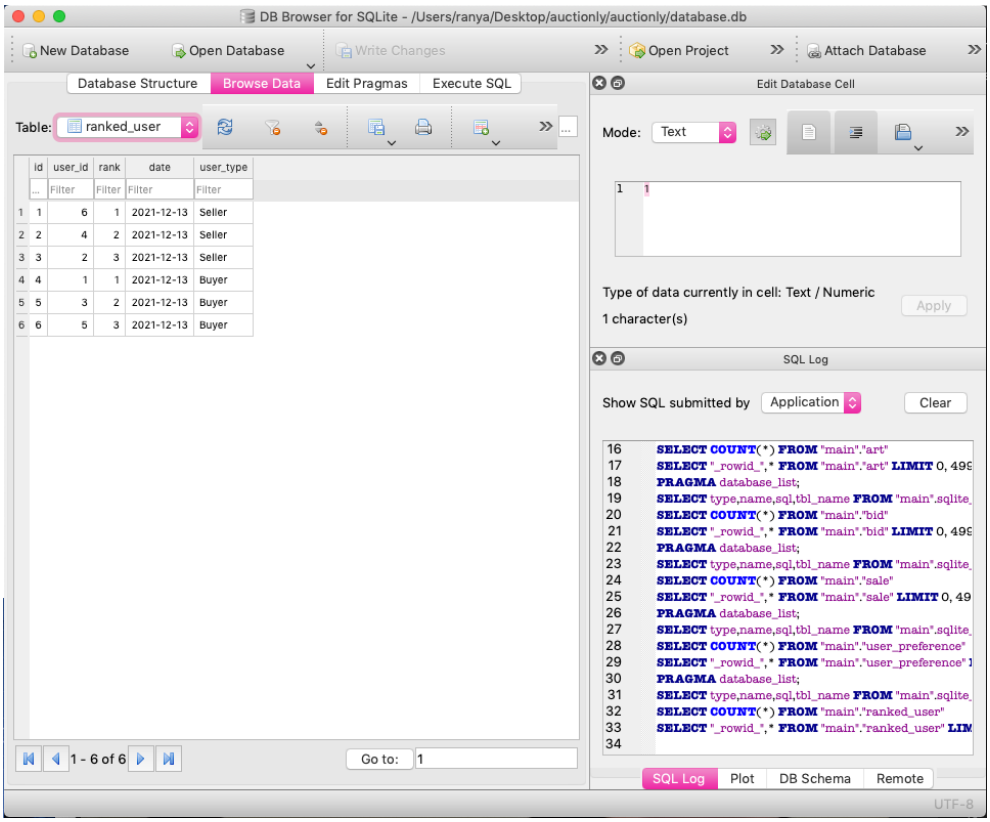
Our database structure in DB browser



Example of data in our bid table



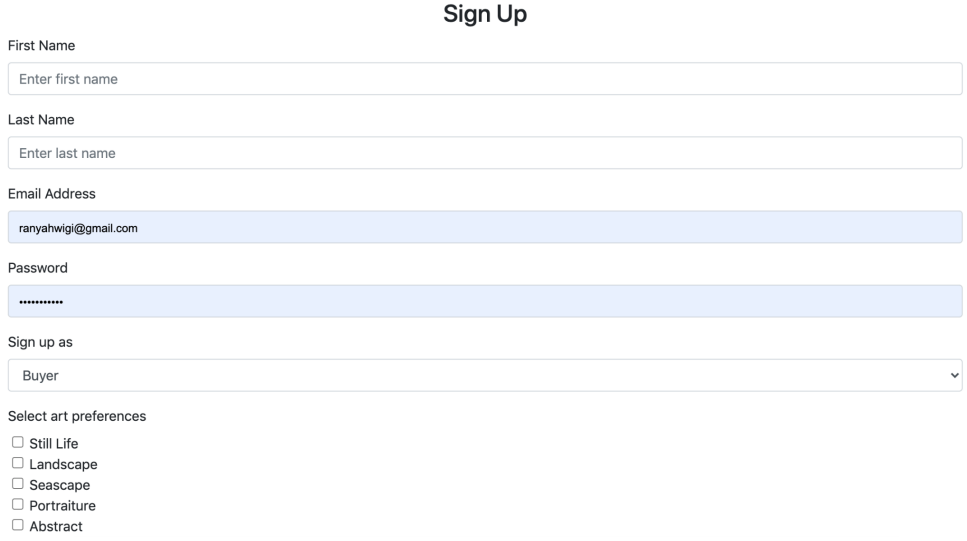
Example of data in our ranked_user table



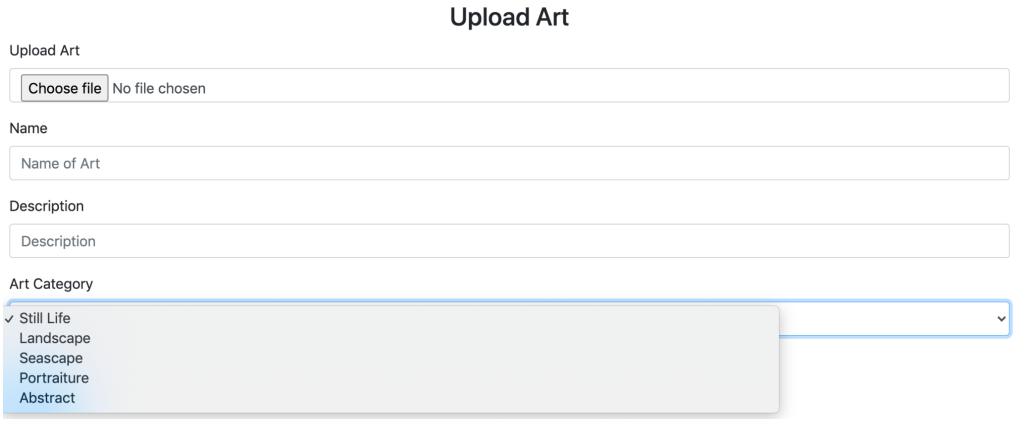
Feed

Our implementation of feed caters to a user's preferences. When a user signs up they can specify art preferences, and when a seller uploads an art piece they can specify what category it falls into so that it's displayed to the correct users.

A user specifies preferences here



A seller specifies the art category here



This is the code that generates the feed according to preferences

```
15      def get_users_feed(self, art_prefs):
16          """returns feed thats catered to the users preferences"""
17          for art in self.all_art:
18              if art.get_art_category() in art_prefs:
19                  if art.get_owner() != self.user_id:
20                      self.art_for_feed.append(art)
21          return self.art_for_feed
```

Rank

We implemented a ranking system for our website that ranks buyers according to how many purchases they make and ranks sellers according to how many sales they make and the highest ranking receive discounts on the commissions that we charge. We perform a new ranking every week and new top rankers are identified. First place receives 30% off, second place gets 20%, and third place is 10% off.

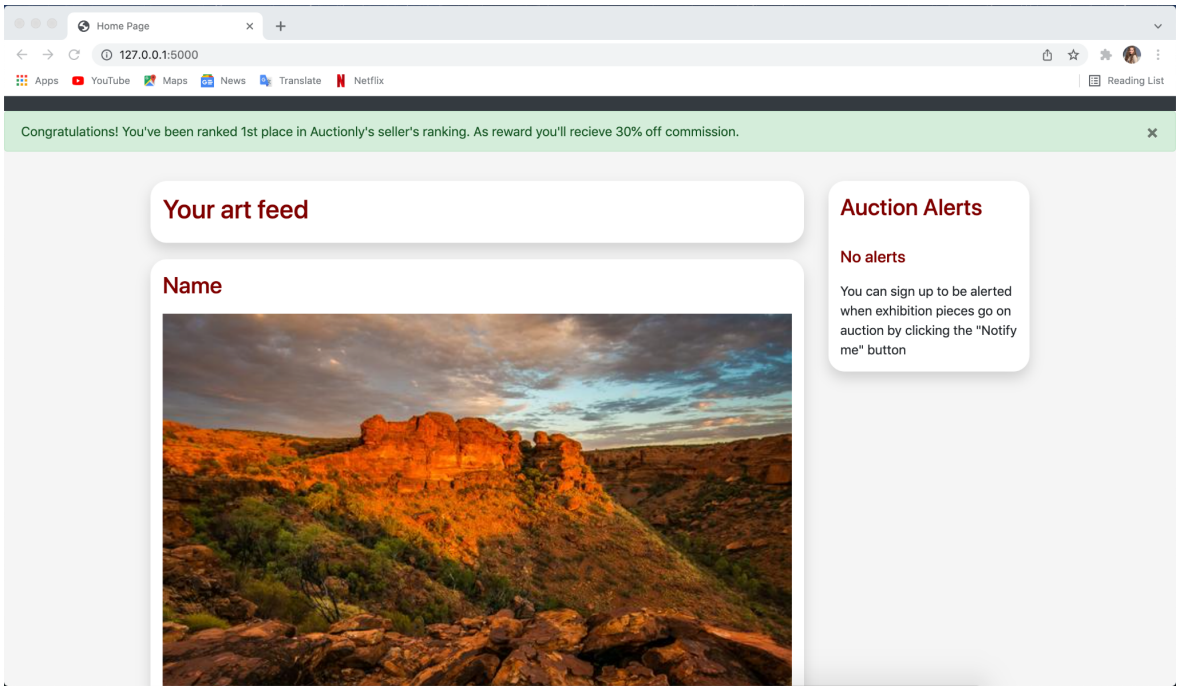
Here is our code for ranking sellers

```
63          #add first three sellers initially
64          for seller in sellers:
65              if(len(top_ranked_sellers)) >= count:
66                  break
67              top_ranked_sellers.append(seller)
68
69          #loop through the rest of the sellers and rank them
70          for seller in sellers:
71              sales = len(Sale.query.filter_by(sold_by=seller.get_user_id()).all()) # sales of current seller in loop
72              sales_1 = len(Sale.query.filter_by(sold_by=top_ranked_sellers[0].get_user_id()).all()) # sales of seller currently in first place
73              sales_2 = len(Sale.query.filter_by(sold_by=top_ranked_sellers[1].get_user_id()).all()) # sales of seller currently in second place
74              sales_3 = len(Sale.query.filter_by(sold_by=top_ranked_sellers[2].get_user_id()).all()) # sales of seller currently in third place
75              if sales > sales_1:
76                  top_ranked_sellers[2] = top_ranked_sellers[1]
77                  top_ranked_sellers[1] = top_ranked_sellers[0]
78                  top_ranked_sellers[0] = seller
79              elif sales_1 > sales > sales_2:
80                  top_ranked_sellers[2] = top_ranked_sellers[1]
81                  top_ranked_sellers[1] = seller
82              elif sales_2 > sales > sales_3:
83                  top_ranked_sellers[2] = seller
```

Here is out code for ranking buyers

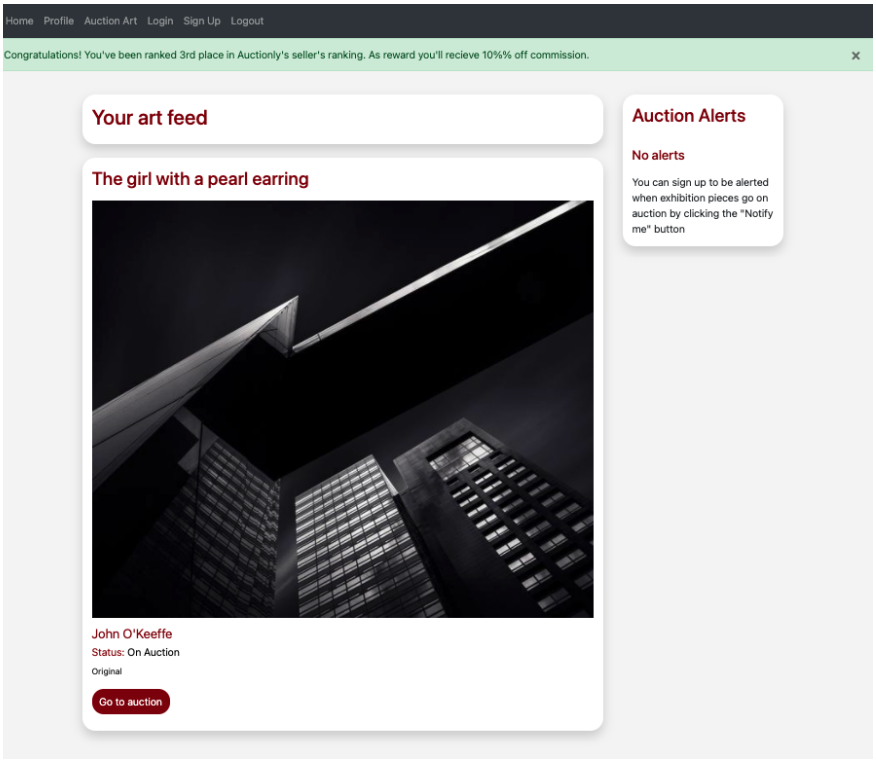
```
85          #add first three buyers initially
86          for buyer in buyers:
87              if(len(top_ranked_buyers)) >= count:
88                  break
89              top_ranked_buyers.append(buyer)
90
91          #loop through the rest of the buyers and rank them
92          for buyer in buyers:
93              buys = len(Sale.query.filter_by(bought_by=buyer.get_user_id()).all()) # buys of current buyer in loop
94              buys_1 = len(Sale.query.filter_by(bought_by=top_ranked_buyers[0].get_user_id()).all()) # buys of buyer currently in first place
95              buys_2 = len(Sale.query.filter_by(bought_by=top_ranked_buyers[1].get_user_id()).all()) # buys of buyer currently in second place
96              buys_3 = len(Sale.query.filter_by(bought_by=top_ranked_buyers[2].get_user_id()).all()) # buys of buyer currently in third place
97              if buys > buys_1:
98                  top_ranked_buyers[2] = top_ranked_buyers[1]
99                  top_ranked_buyers[1] = top_ranked_buyers[0]
100                 top_ranked_buyers[0] = buyer
101             elif buys_1 > buys > buys_2:
102                 top_ranked_buyers[2] = top_ranked_buyers[1]
103                 top_ranked_buyers[1] = buyer
104             elif buys_2 > buys > buys_3:
105                 top_ranked_buyers[2] = buyer
```

They receive the following notification if they were ranked amongst top 3

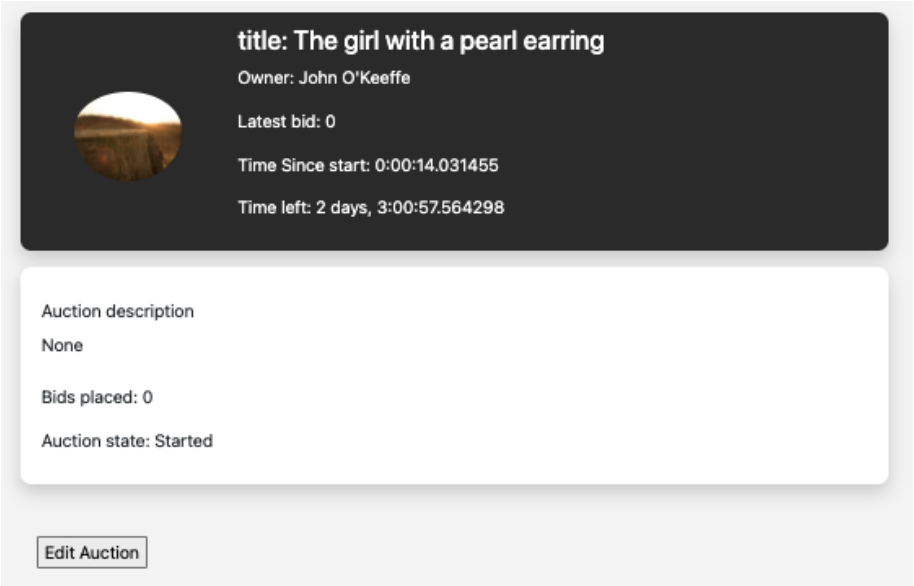


Graphical User Interface / User Interface

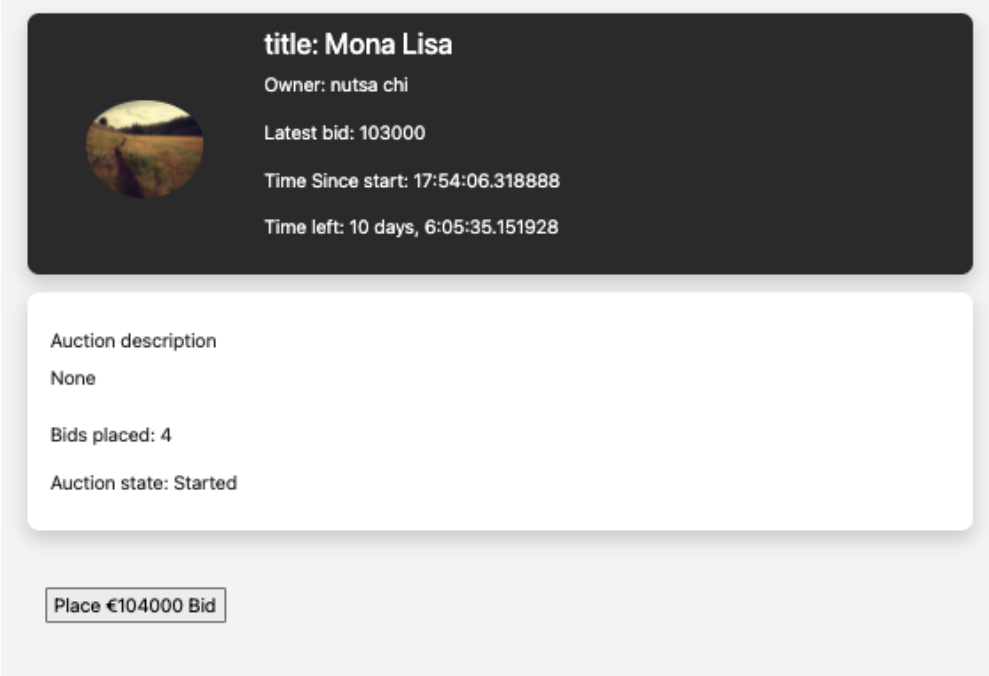
Home feed where the auctionable pieces of art are displayed.



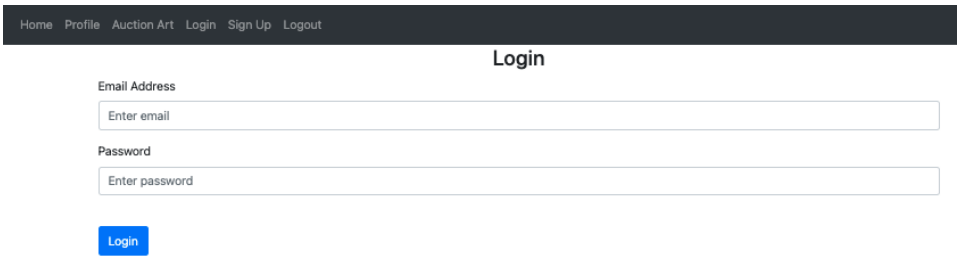
Viewing your own auction. You are able to edit the auction from this view.



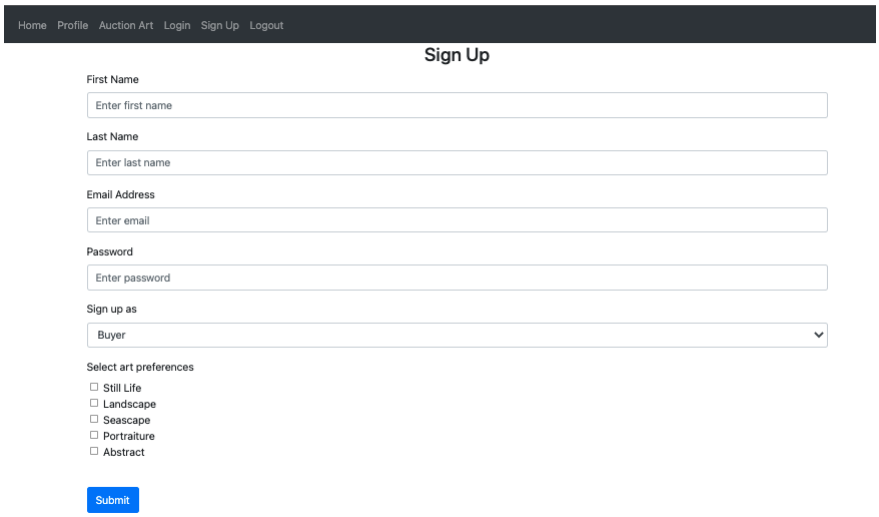
Viewing someone else's auction. You are able to place bids from this view.



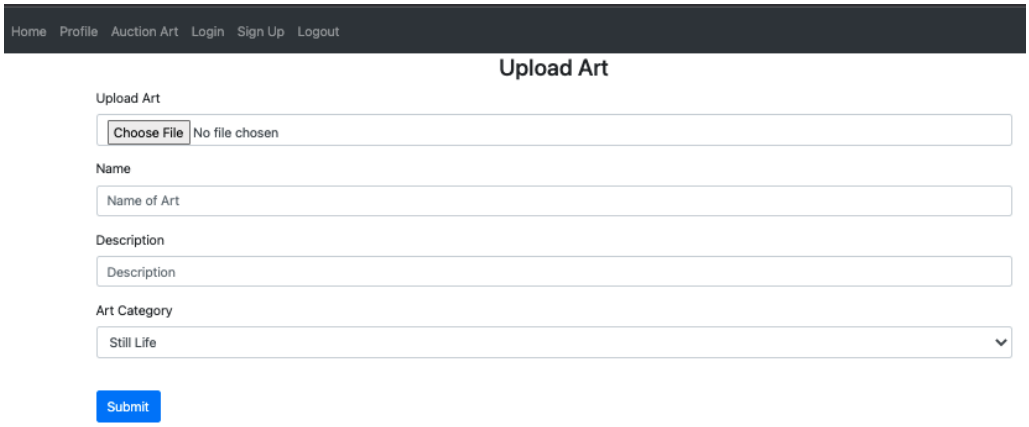
Logging into your account.



Registering an account.



Uploading art



Putting up uploaded art as an auction.

Test Driven Development

```
class TestPayment(unittest.TestCase):
    """ class to unit test the payment system """
    def test_unfreezing(self):
        """ unfreezing the payment and checking that it matches with the expected amount """
        payment = Payment(user_id=1, auction_id=1,
                          time=0, amount=100, bid_id=1)
        payment.unfreeze_payment()
        amount_after_unfreeze = payment.get_amount()
        expected_amount_after_unfreeze = 0
        self.assertEqual(expected_amount_after_unfreeze, amount_after_unfreeze)
```

Test-Driven Development is one of the concepts of extreme programming and allows developers to continuously test their system against some existing test cases to ensure that no added functionality or code breaks any existing functionalities. We created some unit tests throughout the implementation process. These unit tests are located under the tests package and test different parts of the system. We focused

mainly on testing the basic behaviour of our system. Later we automated this process, which you can read more about under the DevOps section of the report.

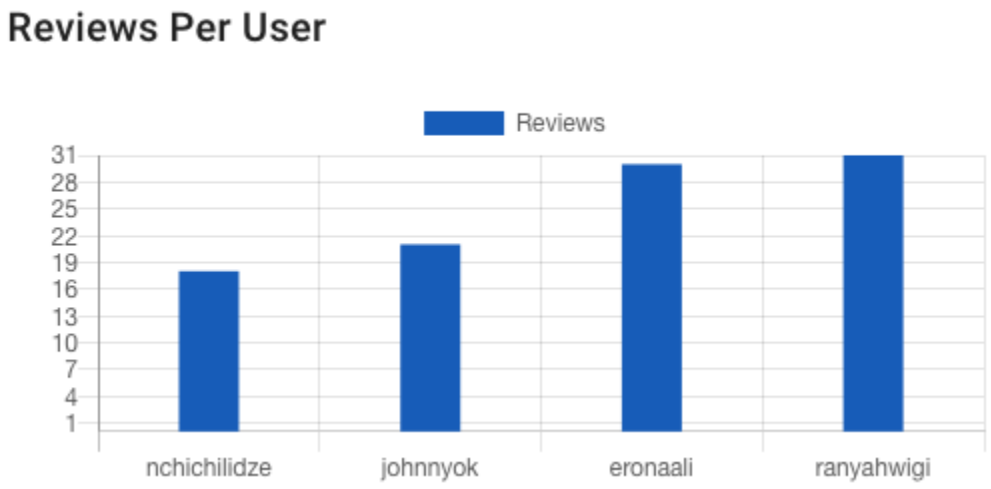
```
def test_buyer_art_prefs(self):
    """ testing the Buyer object art preference setting """
    name = "Nutsa"
    last_name = "CH"
    email = "test@gmail.com"
    password = "1234"

    buyer = Buyer(name, last_name, email, password)
    input_art_prefs = ["Modern", "Contemporary"]
    buyer.set_art_prefs(input_art_prefs)
    self.assertEqual(buyer.get_art_prefs(), input_art_prefs)
```

```
class TestUser(unittest.TestCase):
    """ class for unit testing the user class"""

    def test_user_init(self):
        """ testing the User object constructor """
        name = "Joh"
        last_name = "Doe"
        email = "test@gmail.com"
        password = "1234"
        user = User(name, last_name, email, password)
        self.assertEqual(name, user.get_name())
        self.assertEqual(last_name, user.get_last_name())
```

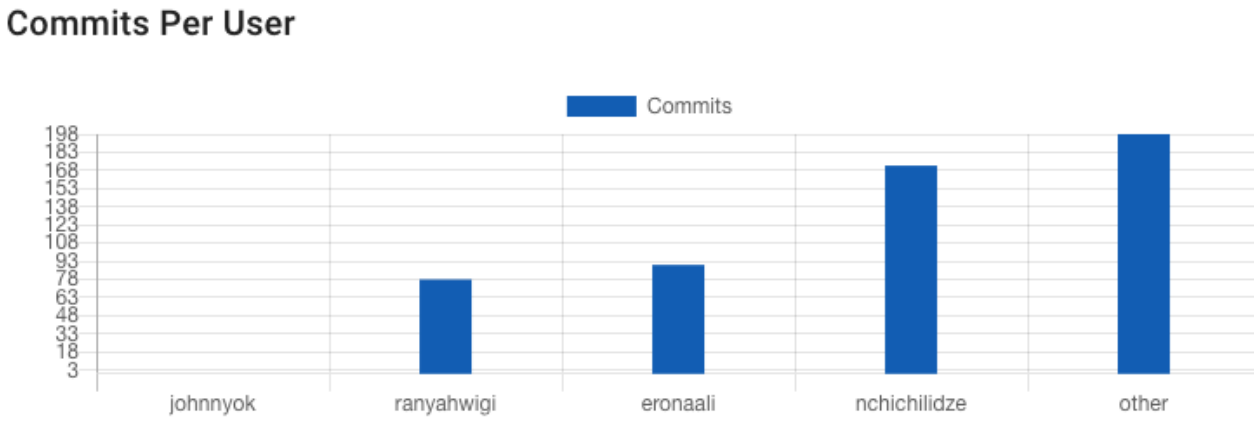
Code - Added Value:

Code Reviews: The Kreoh Platform

We opted in to utilise Kreoh in order to review one another's code during implementation. We connected The project Kreoh account to the GitHub account, which automatically sent any changes to Kreoh to be reviewed. We encouraged each other to leave honest and constructive feedback on the code we were pushing onto the project to make sure that everyone's input and opinions were being considered during implementation, to ensure that the project was meeting everyone's expectations.



Unfortunately Kreoh is still in its Beta phase - which was evident from some of the issues we faced while working with this platform. In many cases, some of our commits were displayed multiple times, or not all. Some of our commits were displayed under the wrong teammates name. The insights feature on the platform wasn't able to correctly estimate the lines of code each team member had committed to the project.



However, the Kreoh team were always responsive and professional from the encounters we had with them. They were always quick to answer any of our queries and even added a feature for resetting your password after we let them know that one of our teammates was not able to remember their password. We appreciate the work they have done on this platform and cannot wait to see what it will look like in its Alpha release.

DevOps: Continuous Integration

GitHub actions allowed us to incorporate continuous integration in our project. We created custom workflows and automated daily running of unit tests, syntactic checks, and dependency management.

Automated Unit Testing with PyTest

PyTest is a python framework that allows developers to unit test complex and scalable systems. It provides different tools to make software testing easier and more efficient. We began by running these unit tests on our own local machines and later managed to automate this process, by using a GitHub actions workflow.

```yaml
name: Unit Testing

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: [3.8]

    steps:
    - uses: actions/checkout@v2
    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v2
      with:
        python-version: ${{ matrix.python-version }}
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install flake8 pytest
        if [ -f requirements.txt ]; then pip install -r requirements.txt; fi
    - name: Test with pytest
      run: |
        python -m pytest tests/tests_users.py
```

After every push onto our repository, the workflow behaves as follows:

1. Sets up Python
2. Installs all the dependencies necessary for our project, these are specified inside the requirements.txt file
3. Runs all the unit tests that are inside the specified file

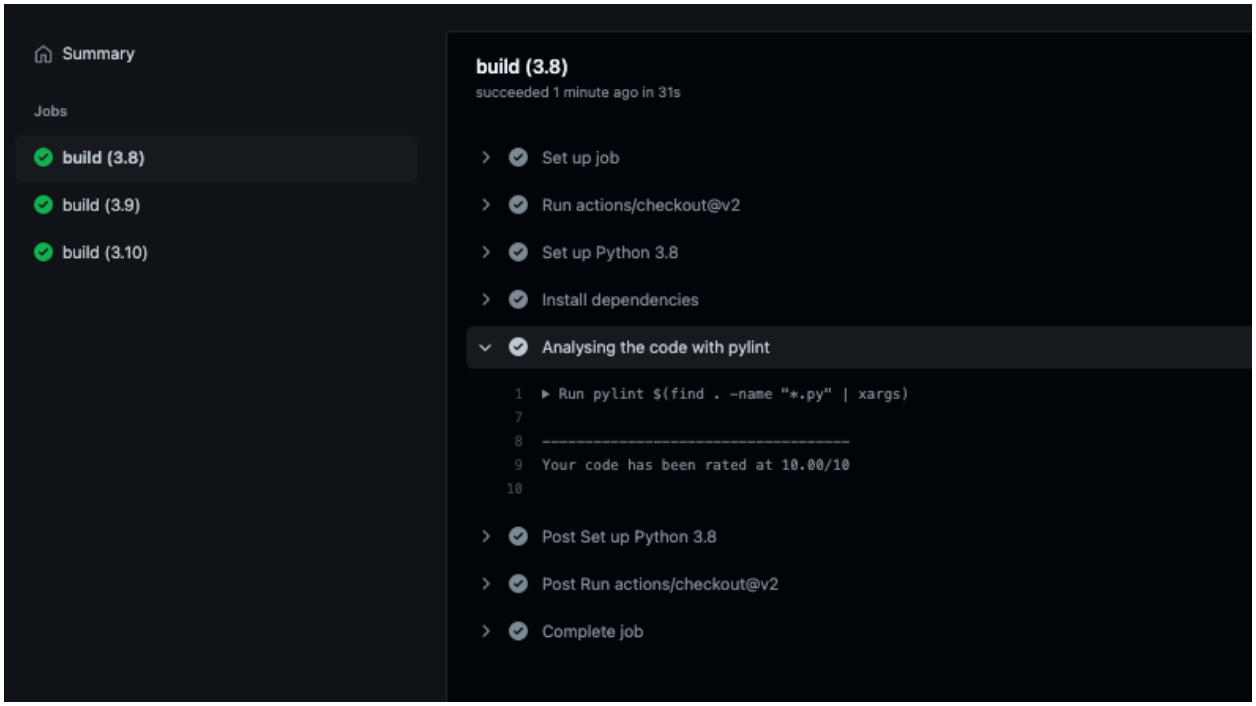
GitHub sends us a notification about whether any of our tests fail for any commit. This allows us to expand our project while continuously checking that our functionalities are intact and no commit breaks our code base.

Automated Dependency Management with Dependabot

Dependabot is a GitHub functionality that we discovered when we were working on populating our requirements.txt file. Due to the fact that our project is both front and back-end, we require a lot of packages to be installed before building our project. We use many different frameworks and packages such as Flask, SQLAlchemy, Jinja, etc. This list grows as we add more functionalities to our project. To ensure extensibility, we incorporated Dependabot within our project, which is a workflow that scans our code base upon every push to the repository, and updates the requirements file in case of identifying new frameworks or packages that are not there yet. We often install something on our machine and forget to update the requirements, therefore Dependabot has made our coding experience much more seamless and efficient.

Automated Syntax Testing with PyLint

The third workflow we run upon every push to our branch is the linting workflow. PyLint is an extremely useful tool that is able to identify any warnings or errors that your compiler might not be aware of. It points out small mistakes such as lines that are too long or extra whitespace after a function declaration, but it is also able to identify bigger issues such as cyclic dependencies and code duplication. After every push onto our repository, PyLint scans our entire codebase and outputs any warnings or errors it finds, as well as giving an overall score to our Python code.



We were able to achieve a score of 10/10 after all of our latest pushes into the repository.

Pair Programming

Pair programming is one of the agile practises that we decided to pay extra mind to. We have frequently tried this practice for our other projects and the results have always been fruitful. One person in the pair writes code, while the other one observes, reviews and critiques it. This helped us teach each other about different parts of software engineering that all of us were not skilled at (front-end development, for example). It also hugely affected our productivity as the "observer" acted as an accountability net.

Facade Design Pattern

Facade can be recognised in a class that has a simple interface but delegates most of the work to other classes.[5] An example of this in our case would be the APIs that we have: Shipment, Payment and Authentication APIs. We have interface classes that delegate most of the work to these APIs. Given that we did not have enough time to implement the APIs, these classes give us the main functionality that we want. However, the idea was to implement the Facade Pattern by creating interfaces to communicate with these APIs.

Shipment

```python
from random import random


class Shipment():

    def ship_art(art_id):
        invoice_number = art_id + random.randint(1, 100000)
        print("The shipment service has received a request for shipment, please refer to your invoice number at" + str(invoice_number))
```

Payment

```python
class Payment(db.Model):
    """this class handles receiving, freezing and sending money through our system"""

    __tablename__ = "payment"

    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey("user.id"))
    auction_id = db.Column(db.Integer, db.ForeignKey("auction.id"))
    time = db.Column(db.DateTime, default=datetime.utcnow)
    amount = db.Column(db.Integer)
    bid_id = db.Column(db.Integer, db.ForeignKey("bid.id"))
    seller_paid = db.Column(db.Integer)
    service_fee_paid = db.Column(db.Integer)

    def __init__(
        self,
        user_id,
        auction_id,
        time,
        amount,
        bid_id,
        seller_paid=0,
        service_fee_paid=0,
    ):
        self.user_id = user_id
        self.auction_id = auction_id
        self.time = time
        self.amount = amount
        self.bid_id = bid_id
        self.seller_paid = seller_paid
        self.service_fee_paid = service_fee_paid
```

Authentication

```python
def authenticate_art(art_id):
    """mocking the authentication process, 1 out of a 100 pieces will come back as fake """
    probability = random.randInt(1000)

    if (probability <= 900):
        print("Art successfully authenticated")
        # update the state of the art
    else:
        print("Art was not authenticated. Came back as fake.")
        # update the state of the art
        # system will continue to block the seller and request a penalty
```

Object Relational Mapping (ORM): SqlAlchemy

ORM is a programming technique in which a metadata descriptor is used to connect object code to a relational database. It converts data between type systems that are unable to coexist within relational databases and OOP languages.

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper which gives application developers the full power and flexibility of SQL.[7] SQL databases behave less like object collections the more size and performance start to matter; object collections behave less like tables and rows the more abstraction starts to matter.[7] SQLAlchemy aims to accommodate both of these principles. It is most famous for its object-relational mapper (ORM), an optional component that provides the data mapper pattern, where classes can be mapped to the database in open ended, multiple ways allowing the object model and database schema to develop in a cleanly decoupled way from the beginning.[7]

So each of our tables was created through a class and whenever an object of that class is created it is added to the database. Extracting a row from the database also results in that row behaving as an object allowing you to run the various methods in the class on it.

Example of table definition through a class

```
11  class User(db.Model, UserMixin):
12      """ initializing the user class and providing getters and setters to
13      interact with the users model in the database.  """
14      __tablename__ = 'user'
15
16      id = db.Column(db.Integer, primary_key=True)
17      email = db.Column(db.String(150), unique=True)
18      password = db.Column(db.String(150))
19      first_name = db.Column(db.String(150))
20      last_name = db.Column(db.String(150))
21      user_type = db.Column(db.String)
22
23      def __init__(self, first_name, last_name, email, password):
24          self.first_name = first_name
25          self.last_name = last_name
26          self.email = email
27          self.password = password
28          self.feed = []
```

The Step Down Rule

In all our classes, we facilitate the step down rule where code is read top down in a clear manner with short methods where possible. E.g. Art class:

```python
"""module containing art class"""
from auctionly import db # pylint: disable=E0401

class Art(db.Model):
    """Art class implemented to create art objects that can be passed between users"""
    # pylint: disable=E1101
    id = db.Column(db.Integer, primary_key=True)
    owner_id = db.Column(db.Integer, db.ForeignKey("user.id"))
    name = db.Column(db.String(150))
    description = db.Column(db.String(150))
    image = db.Column(db.String(150))
    art_category = db.Column(db.String(150))
    up_for_auction = db.Column(db.String(5))

    def __init__(self, name, owner_id, digital_image_path, description, art_category):
        """creates an art object"""
        self.name = name
        self.owner_id = owner_id
        self.image = digital_image_path
        self.description = description
        self.up_for_auction = "False"
        self.art_category = art_category
        self.art_status = "With owner"

    def get_name(self):
        """returns name of the art"""
        return self.name

    def get_owner(self):
        """returns the owner of the art"""
        return self.owner_id
```

DRY (Don't Repeat Yourself)

We made our code as DRY as possible in that we do not repeat any unnecessary code throughout the project. We respect the SOLID rules that in turn result in DRY code.

The Law of Demeter

Given that we have a well decoupled system, we respect that a module should not know about the innards of the objects it manipulates and it should only communicate with close classes (avoiding long call chains) e.g. art.get_name() is short and concise rather than having something like art.get_art().get_info().get_name(). This is also known as a trainwreck and should be avoided as much as possible.

Vertical Formatting

We follow vertical openness where we have a set amount of spaces between imports, packages, classes, methods etc. as well as vertical density where we have lines that are associated together close to one another. E.g. these two methods are close as they are relevant to one another:

```python
def get_sold_by(self):
    """returns who the seller is"""
    return self.sold_by

def get_bought_by(self):
    """returns who they buyer is"""
    return self.bought_by
```

Recovered architecture and design blueprints

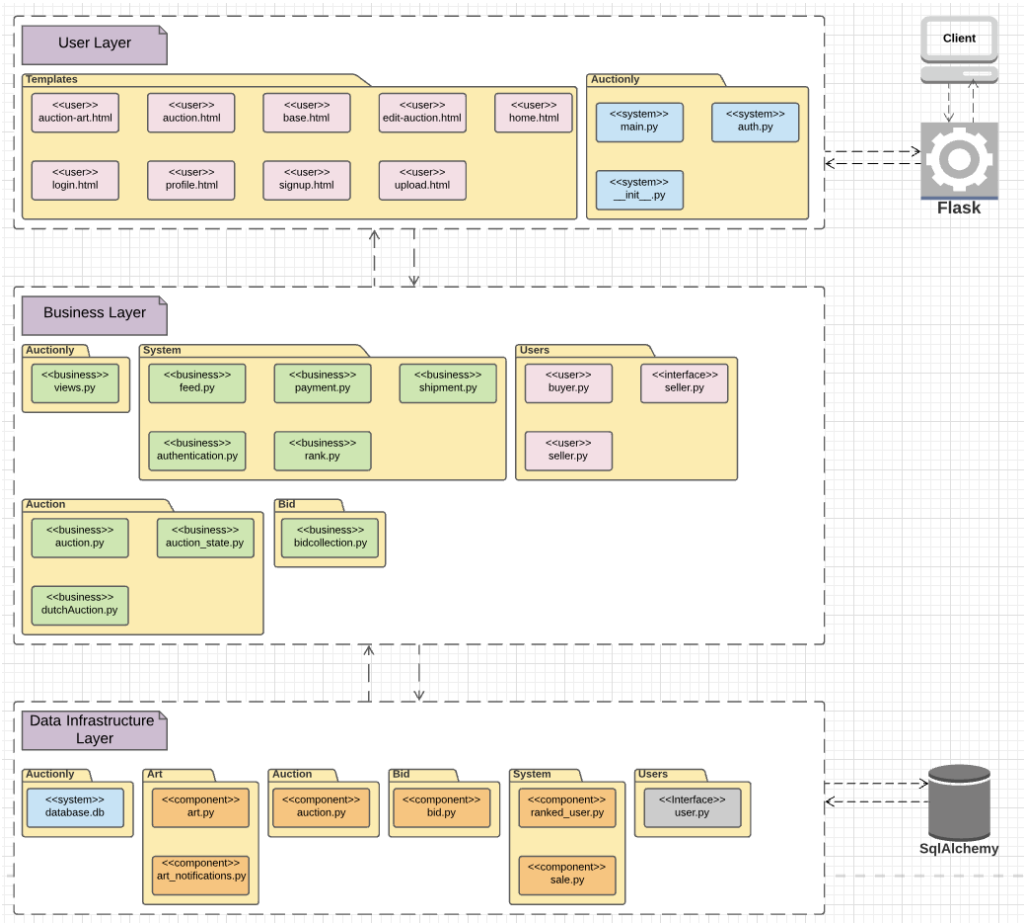
UML workbench

When deciding what UML workbench to go with we were trying to pick between Lucidchart and Microsoft Visio. We decided to go with Lucidchart for the following reasons:

- It's easier to use. We wouldn't need to look up tutorials just to try and implement a simple diagram.[3]
- It's not limited to a specific OS like Microsoft Visio is limited to Windows. As two of us on the team use MacBooks we didn't want to limit the development of the diagrams to the two windows users on the team.[3]
- It's better for collaboration as it allows real time collaboration.[3]

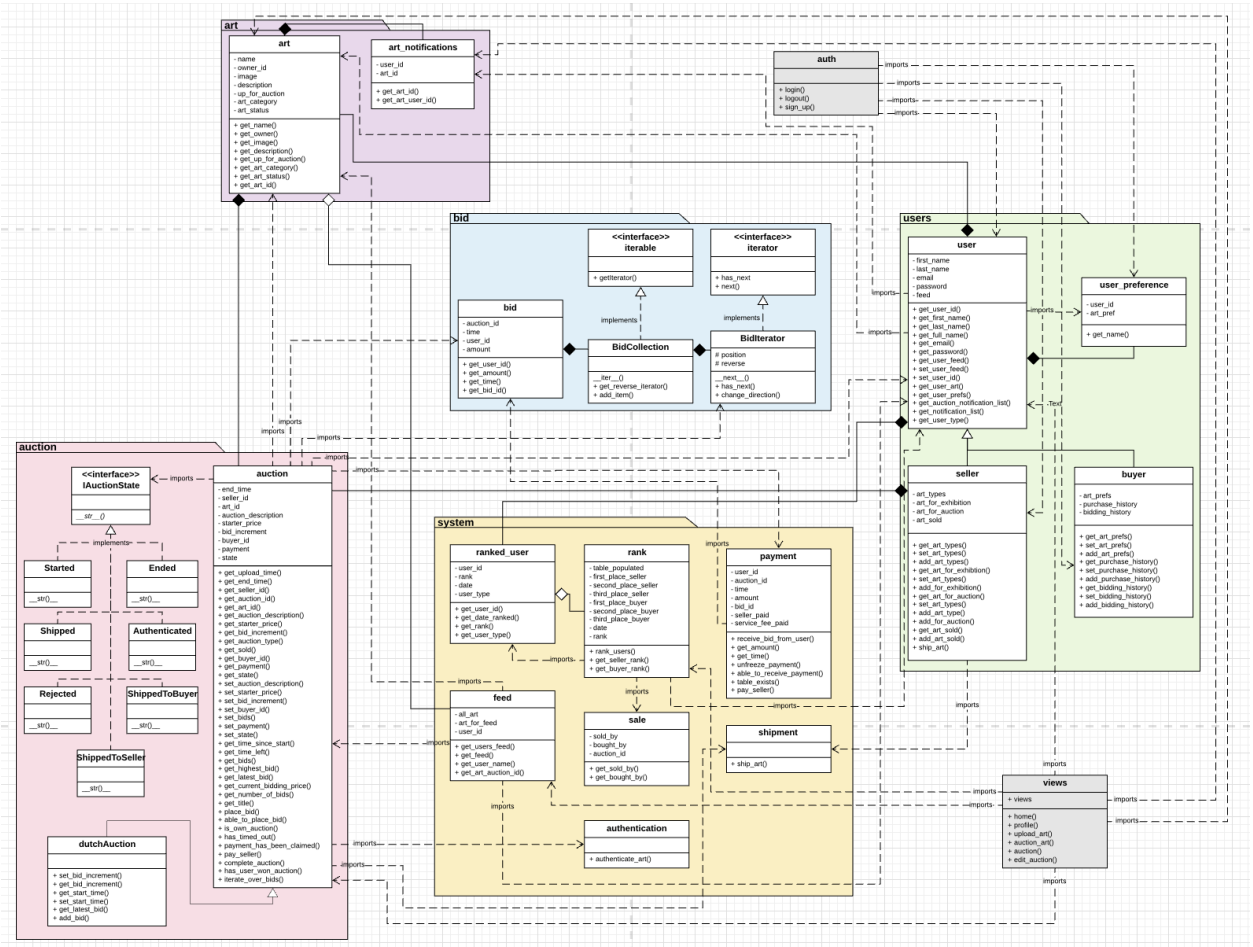
Architectural diagram

The architecture of our system is made up of three layers:
User layer: the packages and classes the user interacts with.
Business layer: the packages and classes that contain our business logic.
Data Infrastructure layer: the packages and classes that implement our database and its tables.
We used the Flask framework to host our website and SqlAlchemy as our database to interact with flask.



Design-time class diagram

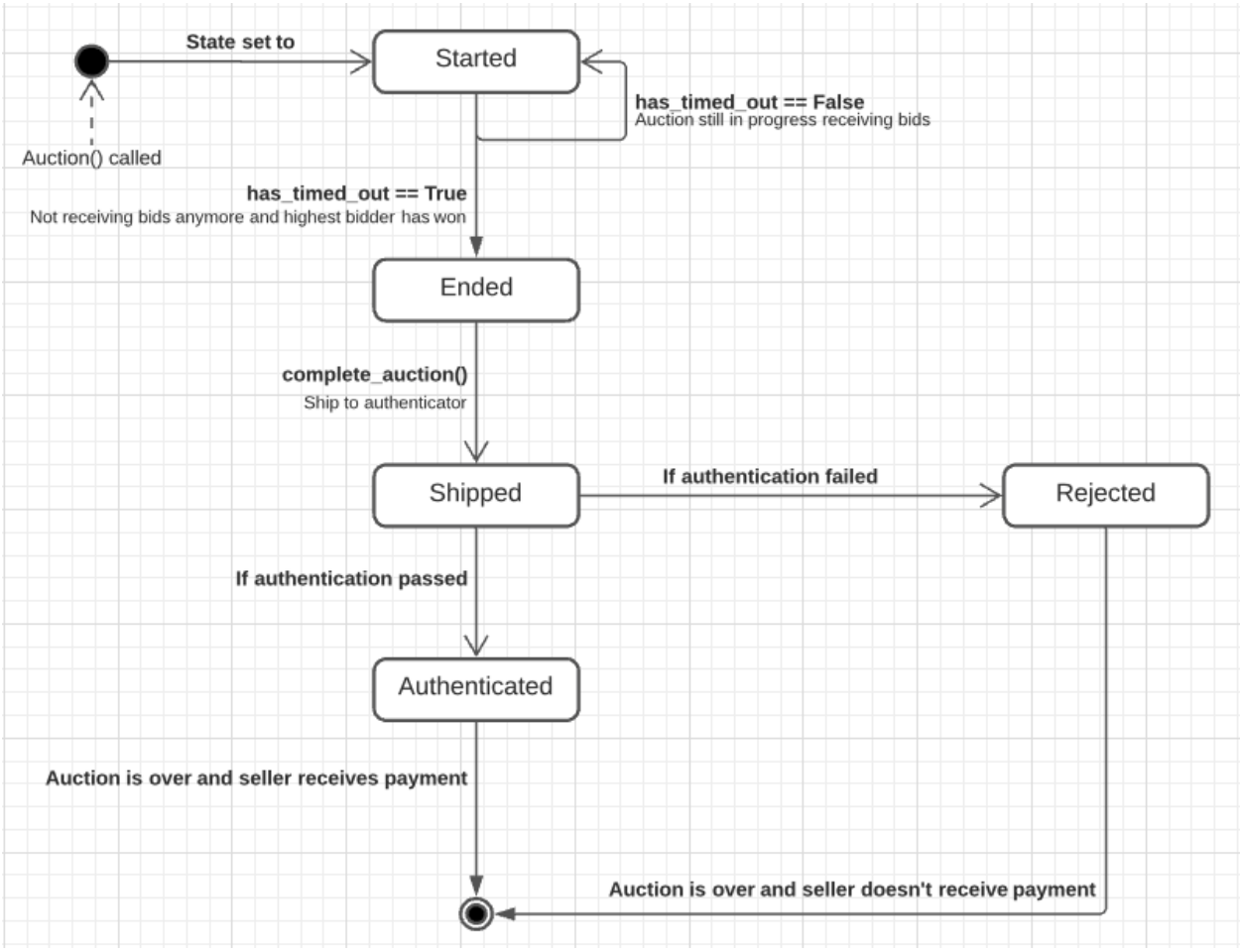
This is the final class diagram that corresponds exactly with our code development.
It's worth mentioning that a lot of the arrows are there because we call a function of a class in another class, so changing the elements of a class doesn't affect the others that import it unless method signatures are changed.



State chart for an Auction object

A state chart is used to describe the different states of a component of a system that are specific to a certain object. Their main purpose is to model the lifetime of an object from creation to termination.[6]

For our state chart we decided to go with describing the states of an Auction object as our communication diagram describes the process of selling an art piece and that requires the construction of an Auction object that changes its state as described below.



Critique

Class diagram

Comparing the class diagrams we can see that initially we expected to have some classes that we didn't need in the end like art category, we ended up implementing this as a list instead of a class because we realised it made more sense that way and made things simpler. We did however implement a user_preferences class which handles storing the user preferences of art categories which we didn't expect we needed at the start.

We can see that we expected to have some classes in different packages than we ended up doing e.g. the APIs package. In the end we had shipment, payment and authentication in the system package alongside rank.

We can also see we expected to create a profile package and have profile classes and subclasses. However, once we began implementation we realised that it was unnecessary to handle all that in python and we could just make use of the database and the front end.

As displayed in the diagram, we expected to develop more functionality than we did, such as a separate rewards system. In the blueprint you can see we just have a rank class which handles the ranking process and then a ranked_user which keeps track of who is ranked. The rewards in this case are just a percentage off which is based on where you land in rank.

Something we would have liked to have the time to include an auction_suggestion package, but from our blueprint you can see we didn't manage to.

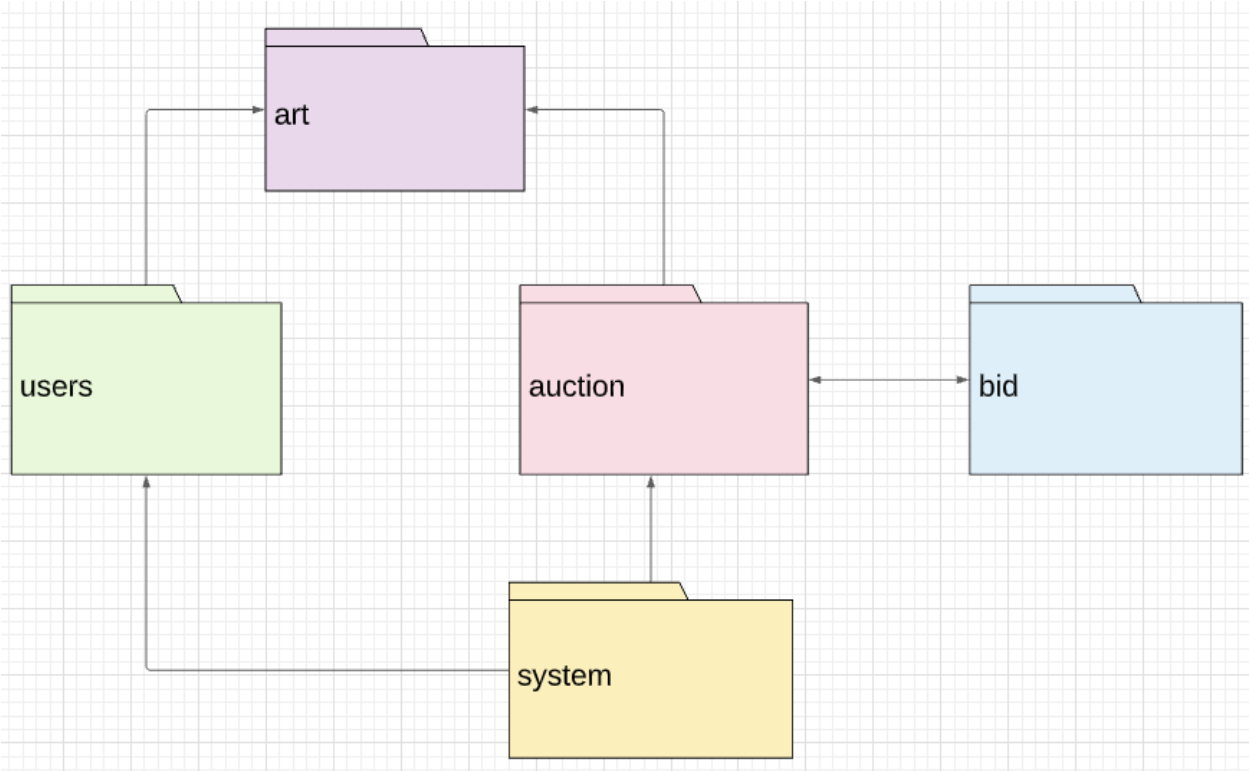
On top of this, we did not consider how bidding would work initially, but as we began developing we knew it needed its own package as each bid is independent of another and has a lot of information associated with it. So you can see in our blueprint we ended up with a bid package, class and some associated classes we didn't expect at the start.

Finally, we expected to have a lot less dependencies than we had. I think we could attribute it to the fact we found it hard to visualise exactly how all the classes and packages would work and use each other. However, looking at the final blueprint we do wish we were able to stick to the small number of dependencies we thought we'd have initially.

Overall, we find the number of arrows we ended up with were quite overwhelming. Looking back, we think with more time we could have handled the number of associations and dependencies better than we did. We focused on coding and getting things working without properly visualising the effect our implementation would have on our class diagram. This has shone a light on how we should focus our attention more on further establishing the class diagram initially in future projects to improve efficiency as well as teamwork overall.

Package diagram

In comparison to the initial package diagram, we can see that we reduced the number of packages as, for example, AuctionSuggestion could be included as a class in the Auction package rather than having it outside of the relevant package, thus resulting in a cluttered system.



Requirements

We developed our requirements twice during the project. We have an initial list of functional requirements and use cases, which is attached in the appendix, that we later changed.
The main issue we had with those requirements is the lack of use cases and also how vague those use cases were. The requirements engineer made the mistake of assuming she knew how to extract use cases after attending the lecture without looking over the lecture again. However, after studying for the midterm it became clear that our use cases had been extracted incorrectly as we didn't have a clear list of who our actors are and the use cases were too few and vague. The method used for use case realisation was to just start drawing a use case diagram which is where our issues arose, so after realising the mistake the requirements engineer went at it again and this time started with a list of actors of the system and then for each actor made a list of use cases and a use case diagram. We ended up with far more use cases which felt more right and more fitting for the system we had in mind. As result we also had better functional requirements that aligned more with our use cases.

Overall Implementation

Overall we are happy with our implementation, however, there are a few changes we would make if we could go back.
We would spend less time on documentation in the early weeks and start coding earlier to give us more time to spend on documentation in the final weeks.
We would read the spec more carefully as initially we thought we had to implement everything and not just a subset of use cases which caused us unnecessary stress and lost time.
We would make our main focus the design patterns and develop our system around them. Because we made our system the main focus and then looked for where design patterns fit which means we ended up with less than we would've liked.
We would've thought more carefully about picking python because we found it a little difficult to understand the way python implements interfaces, abstract classes and methods, and inheritance. Perhaps using a language that we're familiar with and how it implements those concepts, like Java, would've been more helpful for us.
Finally, we would've abandoned the idea of a website and just went with a simple GUI system instead.

References

[1] Ball & Socket: https://www.lucidchart.com/pages/uml-component-diagram
[2] Component Diagram- https://online.visual-paradigm.com/diagrams/tutorials/component-diagram-tutorial/
[3] UML workbench - https://www.lucidchart.com/blog/reasons-why-lucidchart-is-the-perfect-microsoft-visio-replacement
[4] Observer design pattern - https://www.tutorialspoint.com/design_pattern/observer_pattern.htm
[5] Facade Pattern - https://refactoring.guru/design-patterns/facade/python/example#:~:text=Facade%20is%20a%20structural%20design,unwanted%20dependencies%20to%20one%20place.
[6] State chart - https://www.tutorialspoint.com/uml/uml_statechart_diagram.htm
[7] ORM - https://www.sqlalchemy.org

Appendix

Old requirements

Initially these were our functional requirements and use cases, later after looking more in depth into the material we realised that we could make a better job of them which is what we included above in the requirements section. We critiqued these requirements and the ones we included in the critique section.

A listing of functional requirements

1. System shall allow artists to register a profile.
2. System shall allow artists to edit their profile.
3. System shall allow registered artists to exhibit their work.
4. System shall allow registered artists to start an auction for their art.
5. System shall allow registered artists to customise auction properties.
6. System shall allow buyers to register a profile.
7. System shall allow buyers to edit their profile.
8. System shall allow registered buyers to specify their art preferences.
9. System shall allow registered buyers to post bids on art up for auction.
10. System shall allow registered buyers to buy art they've won bidding on.
11. System shall allow anyone to view art pieces up for bid.
12. System shall suggest auction properties to artists when putting their pieces up for auction.
13. System shall rank artists by popularity.
14. System shall provide commission discounts to top performing sellers.
15. System shall monitor auction progress and offer suggestions to the artist on how to proceed.
16. System shall display buyers bought pieces on their profile.

17. System shall rank buyers by their activity.
18. System shall offer special discounts to top buyers.
19. System shall display how much money an artist has made to that artist.
20. System shall allow users to disable their accounts.
21. System shall maintain a log of transactions between sellers and buyers.

Use cases

- Publish art for auction
- Create personal profile
- Post bids
- Buy art
- Specify art preferences
- View art pieces

Project Links

- GitHub repository
- Trello Kanban Board
- Kreoh Code Reviews

Weekly Diary [Weeks 3 - 15]

Week 3	
Ranya	
Contribution this week	Brainstormed with the team, read through some previous projects, and took on the project manager, requirements engineer and designer roles.
Issues arising this week	None
Plan for next week	Start the requirements
Nutsa	
Contribution this week	Created a trello board to ensure productivity. Familiarised myself with the assignment. Set up the project document. Researched different roles in the software model and picked the ones I found fitting for myself.
Issues arising this week	None
Plan for next week	Pick a project scenario.
Erona	
Contribution this week	Brief overview of project outline from lab. Chose roles I thought I could work best in. (Systems Analyst, Documentation Manager)
Issues arising this week	None
Plan for next week	Started having a small look into Systems Analysis (Clean Code ch 11)

Johnny	
Contribution this week	Familiarised myself with the trello board. Created and set up the project Figma. Worked with the team to choose roles that we thought were best fit for each other.
Issues arising this week	None
Plan for next week	Work on GUI prototypes

Week 4	
Ranya	
Contribution this week	Synced with team members on work that needs to be done and agreed on tasks for the week. Began thinking of requirements
Issues arising this week	
Plan for next week	complete requirements and start programming
Nutsa	
Contribution this week	Researched different projects and provided ideas for a project scenario. Brainstormed with the team and picked the idea of an online auctioning service.
Issues arising this week	None
Plan for next week	Begin more detailed design of the project.
Erona	
Contribution this week	Brainstormed with the team for different project ideas and chose one together.
Issues arising this week	
Plan for next week	Settle with an idea, look more into systems analysis for next week.
Johnny	
Contribution this week	Investigated different potential projects for the team and chose one together. Worked on GUI prototypes for the landing page, registering and the home feed page on Figma.
Issues arising this week	None
Plan for next week	Research technical implementation and tools

Week 5	
Ranya	
Contribution this week	Continued with requirements and worked on thinking up more business rules for our system
Issues arising this week	Developing requirements took longer than anticipated.
Plan for next week	Begin coding
Nutsa	

Contribution this week	Worked on identifying the possible classes and methods with the team, listed them out and brainstormed on how they might interact
Issues arising this week	None
Plan for next week	Work on users package
Erona	
Contribution this week	Since I volunteered to be the systems analyst, I have been working on research online as well as going through lectures and providing readings on how to build a robust system that will satisfy our expectations for the website.
Issues arising this week	
Plan for next week	Complete systems analysis aspect of the project -> class model
Johnny	
Contribution this week	Researched what tools to use for the classes, evaluated different libraries to use, and what tools to use for DevOps. Discussed these solutions with the team and came to an agreement.
Issues arising this week	None
Plan for next week	Start coding

Week 6	
Ranya	
Contribution this week	Began coding some classes for the system (art class), as was suggested to us, to get a better idea of business rules for our system.
Issues arising this week	wasn't happy with our requirements
Plan for next week	continue coding
Nutsa	
Contribution this week	Was working on the users package, as well as drawing a package diagram for the project documentation
Issues arising this week	Hard to create the package diagram before we have much coding done
Plan for next week	Continue with coding iterations
Erona	
Contribution this week	Worked on the Analysis Class Diagram
Issues arising this week	Struggled with identifying relationship between classes
Plan for next week	Finish the diagram and leave it in the past - look at it once project is done, looking at use cases next week
Johnny	
Contribution this week	Began setting up the code for our tech stack. Followed tutorials on how to setup a flask project to use for our system.

Issues arising this week	None
Plan for next week	Continue coding

Week 7	
Ranya	
Contribution this week	redone requirements as I felt they didn't reflect our system correctly and the use cases didn't seem like they were enough or done right.
Issues arising this week	none
Plan for next week	forget about requirements and focus on coding
Nutsa	
Contribution this week	Refactoring some code in the users package. Starting to plan how to implement some concrete use cases.
Issues arising this week	Not much time left.
Plan for next week	Continue coding
Erona	
Contribution this week	Tidied up the Analysis Class diagram, left it in past now
Issues arising this week	
Plan for next week	Start looking into Kreoh, tying it in with out repo and make some commits to test it out
Johnny	
Contribution this week	Setup the authentication system for being able to login and logout and handle sessions.
Issues arising this week	None
Plan for next week	Continue with the coding

Week 8	
Ranya	
Contribution this week	Looking into SqlAlchemy as a database that we can use for our website and made a list of tables and their schema for storing our data
Issues arising this week	having trouble getting it to work on my laptop
Plan for next week	get the database working and code the use cases assigned to me which were displaying user feed and rank users
Nutsa	
Contribution this week	Working on mocking the auction process in the __init__class as well as finishing up the classes I've. started working on
Issues arising this week	Not much time left so trying to bring everything together

Plan for next week	Continue with coding iterations
Erona	
Contribution this week	Set up github repo on Kreoh, made a few initial commits wrt Flask
Issues arising this week	
Plan for next week	Sort out Stack to use - choosing which DB works best, implement 4 use cases
Johnny	
Contribution this week	Worked on trying to finish the base code for the flask integration, cleaning up the code to make sure there was no errors and everything worked.
Issues arising this week	None
Plan for next week	Continue coding
Week 9	
Ranya	
Contribution this week	coded the system package which contains the user feed, ranking system
Issues arising this week	Some trouble on implementing the code so that it works with flask
Plan for next week	get the GUI set up so I can make sure the feed is loading as I would like and ranked users are notified
Nutsa	
Contribution this week	Mocked the shipment and payment classes. They are mock APIs and are easily integrated with other classes.
Issues arising this week	None
Plan for next week	Focus more on the front-end.
Erona	
Contribution this week	code reviews on kreoh, learning to use it
Issues arising this week	
Plan for next week	look into tier architectures
Johnny	
Contribution this week	Worked on on integrating the SQL alchemy module into the current classes to save the variables in the database, so we don’t have to create new objects each time we run our code
Issues arising this week	None
Plan for next week	Continue coding
Week 10	
Ranya	

Contribution this week	coded user preferences, frontend feed, frontend ranking, integrating the frontend and backend logic of my assigned use cases.
Issues arising this week	Having trouble with the integration as im worried this is becoming very frontend heavy instead of backend
Plan for next week	Finish coding and move onto to my designer duties
Nutsa	
Contribution this week	Worked on the front-end of displaying an auction. The user is able to view details about the auction as well as place a bid. Placing a bid is stored in the database and the system is able to fetch bidding information from the database.
Issues arising this week	We are spending a lot of time on front end design because we want our project to be complete but struggling to move on and spend more time on design principles.
Plan for next week	Continue working on the Bid and Auction classes.
Erona	
Contribution this week	continued with coding the profile and doing code reviews to make sure im on the same page as my teammates
Issues arising this week	struggled with code getting convoluted
Plan for next week	continue coding if necessary and refer to initial analysis class diagram
Johnny	
Contribution this week	Worked on the art class to be able to upload art, and worked on the auction class to be able to auction the art and save it the the database.
Issues arising this week	None
Plan for next week	Continue coding
Week 11	
Ranya	
Contribution this week	Continued with coding and implementing the ranking system to work completely from the database without user input.
Issues arising this week	code is starting to look messy
Plan for next week	finish coding and clean up code
Nutsa	
Contribution this week	Working on the auction class. Adding the ability for the user to edit an existing auction. Built the payment system which is able to freeze/send/receive money. It is tied to a payment database that keeps track of all these transactions.
Issues arising this week	We have a lot left to do.
Plan for next week	Continue working on the auction class.
Erona	

Contribution this week	continued with coding
Issues arising this week	struggling to understand some concepts in our project - looking into them more and having discussions with my team
Plan for next week	pull all changes, look at all the code and try to understand how its functioning right now as well as seeing if i can tidy up some stuff
Johnny	
Contribution this week	Continued coding and made fixes to the code
Issues arising this week	None
Plan for next week	Continue working
Week 12	
Ranya	
Contribution this week	Continued with coding and cleaned up code.
Issues arising this week	feeling like we're behind on the report
Plan for next week	focus solely on the report
Nutsa	
Contribution this week	Created some sample unit tests and am planning to build a daily workflow as a part of our continuous integration.
Issues arising this week	There's a lot of code I found that is redundant and sometimes confusing since we started the project 10 weeks ago. Need a lot of deletion.
Plan for next week	Work on continuous integration.
Erona	
Contribution this week	continue with code and cleanup, looking into report to try and see what we need to do and what we have completed
Issues arising this week	feeling behind on report
Plan for next week	focus on report and pylint
Johnny	
Contribution this week	Continued coding classes, editing classes, and started trying to think about how to implement use cases.
Issues arising this week	None
Plan for next week	Focus working on the project
Week 13	
Ranya	
Contribution this week	Research into design patterns through studying for the midterm and noting down where I found them in our code. I also began researching the blueprints I need to develop.

Issues arising this week	none
Plan for next week	Work on blueprints.
Nutsa	
Contribution this week	Spent most of my time preparing for the final exam. Revising for the exam helped me think of different design patterns and principles that we can incorporate for our project.
Issues arising this week	Time's almost up and we still have many ideas for the future.
Plan for next week	Implement the iterator design pattern, look at linter.
Erona	
Contribution this week	code cleanup, learning more about design patterns and good code practises from Clean Code
Issues arising this week	
Plan for next week	try to apply SOLID to code
Johnny	
Contribution this week	Cleaned up code, focused on studying for exams, and researched design patterns to implement into the project
Issues arising this week	None
Plan for next week	Work on finishing the project

Week 14	
Ranya	
Contribution this week	Fixing pylint errors in the code I believed and drawing up the blueprints of the system.
Issues arising this week	Finding it hard to put into words why we made things the way we did.
Plan for next week	Final bits and pieces and submit the project.
Nutsa	
Contribution this week	Working on the report and refactoring the code I have written in the past.
Issues arising this week	Some of the code is over 1 month old and is difficult to sort what's necessary and what needs to be deleted.
Plan for next week	Finish up the report and code.
Erona	
Contribution this week	Apply SOLID to code use pylint to tidy up any mishaps
Issues arising this week	
Plan for next week	Wrap up - tidying and finalising the report
Johnny	
Contribution this week	Worked on finishing the purchase art use case, and fixing any errors that arised with the project.

Issues arising this week	None
Plan for next week	Finish the project

Week 15	
Ranya	
Contribution this week	Finalised blueprints by updating according to final code. Worked on final bits of report and brought it all together.
Issues arising this week	None.
Plan for next week	Party :)
Nutsa	
Contribution this week	Finalised the report, worked on fixing some linting errors for the classes I had made. Added an implementation of the iteration design pattern. A lot of refactoring.
Issues arising this week	None.
Plan for next week	None.
Erona	
Contribution this week	Finalising the report with team, ensuring diagrams are up to standard and that presentation is up to standards as well as helping finalise some code
Issues arising this week	
Plan for next week	Hibernate :) ✌️
Johnny	
Contribution this week	Finalised the report, made sure there were no errors with the code, implemented the state design pattern, and worked to make sure DevOps was passing.
Issues arising this week	None
Plan for next week	