University of Messina

Department of Mathematics, Computer sciences, Physical sciences and Earth sciences (MIFT)

Data Analysis course in INFORMATICS

AUCTIONS

A.A 2023/2024

Professor: Salvatore Distefano

Name: Bhavesh Baviskar

Introduction

Our Auction Website is an innovative platform designed to facilitate online auctions, allowing users to create accounts, log in, and place bids on a variety of items. The website features a seamless user experience where participants can engage in auctions, monitor their bids, and make transactions securely. It integrates with Ethereum wallets via Third Web smart contracts, enabling blockchain-based transactions for enhanced security and transparency.

The website is deployed using Firebase Hosting, ensuring fast and secure content delivery. User authentication and data storage are managed by Firebase Authentication and Cloud Storage, providing robust and scalable backend support. By leveraging these advanced technologies, our platform offers a reliable and user-friendly environment for online auctions, combining the benefits of blockchain and cloud services.

Software Process

1. Chosen process: Scrum

Scrum is one of the most popular frameworks for Agile implementation. Scrum focuses on managing iterative development rather than specific Agile practices.

2. Why Scrum?

As I say in the first section, Scrum is an Agile framework that is particularly suited for complex projects with rapidly changing requirements, which is often the case in software development. In this project, requirements change based on recommendation systems research is very likely to happen, so Scrum or Agile in general is a reasonable choice for this project. Additionally, Scrum allows for frequent iterations and feedback loops, which can help ensure that the final product meets the needs of stakeholders.

Mention of stakeholders, because this project only has the participation of one person, traditional Agile will not be suitable because all the members in the team will share all roles, all responsibilities. With Scrum, there is a clear division of roles in the three positions of Product Owner, Scrum Master and Development Team which makes it easier to simulate a team using this methodology. I can still understand the difficulty of each position in the project through the work that role needs to do.

Furthermore, Scrum can help ensure that the project remains on track and that potential issues are identified and addressed early on in the development process. Scrum’s emphasis on frequent communication and collaboration can help ensure that all members of the team are aware of any potential issues and are working together to address them.

Overall, choosing Scrum can help ensure that the final product meets the needs of stakeholders, is delivered on time, and is of high quality.

3. Three phases in Scrum

There are three phases in Scrum:

- Initial phase: in this phase we establish the general objective for the project and design the software architecture. This phase includes the processes related to initiation of a project: Create Project Vision, Identify Stakeholder(s), Scrum Master and Product Owner, Form Scrum Team, Create Prioritized Product Backlog, and Conduct Release Planning.

- Sprint cycle phase: this is a series of sprint cycles, where each cycle develops an increment of the system. This phase includes the processes related to implementation of a project: Create Deliverables, Conduct Daily Standup, Groom Prioritized Product Backlog, and Deliver Deliverables.

- Project closure phase: wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project. This phase includes the processes related to releasing a project: Ship Deliverables and Retrospect Project.

4. Three main roles in Scrum

There are three main roles in Scrum:

- Product Owner: The Product Owner is responsible for maximizing the value of the product resulting from the work of the Development Team. They are also responsible for managing the Product Backlog. They do analysis, are business-oriented, do external interactions-interface. They will keep the team Build the right thing.

- Scrum Master: The scrum master is responsible for ensuring that Scrum is understood and enacted by all team members and stakeholders. They are servant leader, advisor, coach, mentor, facilitator. They will keep the team Build it fast.

- Development Team: The development team is responsible for delivering a potentially releasable increment of “Done” product at the end of each Sprint. They are selfmanagement or self-organizing. They will keep the team Build things right.

5. Sprint cycle

Sprint are fixed length, normally 2-4 weeks. In this project one sprint is 5 working days. Usually, each sprint will have 4 phases:

- Planning: before each sprint, the team selects what it will commit to delivering by the end of the sprint, starting at the top of the product Backlog. During the sprint, what the team committed to deliver does not change, and the end-date of the sprint does not change. If something major comes up, the Product Owner can direct the team to terminate the Sprint prematurely and start a new one.

- Implementation – Daily Scrum Meeting: each day, the team has a short meeting to update each other on progress and surface blocks. They stand up, to keep it fast. To keep the meeting to less than 15 minutes, everyone reports just 3 things: done since yesterday, done by tomorrow, and blocks. Burndown Chart are necessary for the team to successfully self-manage.

- Review: at the end of the sprint, the Product Owner, Team, Scrum Master, and Stakeholder(s) come together and see a demo of what the team has produced. The Product Owner gathers feedback from everyone on ways to improve what has been built. Review of the timeline, budget, potential capabilities, and marketplace for the next anticipated releases of functionality and capabilities.

- Retrospective: Team, Product Owner, and Scrum Master meet at the end of each Sprint to review their way of working, to improve effectiveness. Usually, it will take 3 hours for one month Sprint. This is the opportunity for the Scrum Team to improve and all members should be in attendance, discussing about: what went well, what could be improved, and what we will commit to improving in the next Sprint.

Initial Phase

1. Project members

- Project Owner: Bhavesh Baviskar
- SCRUM master: Abdul Raqeeb
- Members in team: Bhavesh Baviskar, Abdul Raqeeb

Taking on all three roles in SCRUM is not recommended because SCRUM and software development processes in general were born to solve problems related to independent interpersonal cooperation. I must do this project alone because I could not find the right partner at the right time. However, I will try to play each role independently so that I can understand in a project what the roles will contribute.

2. Project ideal

The Ideal of Project is a website where users can bid for the bids they might like. The website is a responsive Software project with different processes contributing for its functioning.

3. Technologies Used

- Git for source control

Git is a popular version control system for tracking changes in software code, supporting collaboration, branching, and merging. It is widely used in software development to manage and organize code changes efficiently.

- Draw.io for Designing

  Draw.io is a free, online diagramming tool for making flowcharts, UML diagrams, network diagrams, and more. It offers a simple interface, pre-made templates and shapes, collaboration, and integration with other tools. All diagrams in this project are drawn with Draw.io .

- Azure DevOps for Scrum management

  Azure DevOps is a suite of development tools, services, and features offered by Microsoft for software development teams to plan, develop, test, and deliver software. It includes services for Agile project management, continuous integration and delivery, testing, and more.

- Cloud Firebase Data store

  Cloud Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud. Like Firebase Database, it keeps your data in sync across client apps through realtime listeners and offers offline support for mobile and web so you can build responsive apps that work regardless of network latency or Internet connectivity. Cloud Firestore also offers seamless integration with other Firebase and Google Cloud products, including Cloud Functions. Authentication, storage, hosting

- Cloud Firebase Authentication

  Firebase Authentication offers backend services, SDKs, and UI libraries for user authentication in apps. It supports various authentication methods, including passwords, phone numbers, and federated identity providers. It integrates well with other Firebase services and uses industry standards like OAuth 2.0 and OpenID Connect, allowing easy integration with custom backends.

- Firebase Storage

  Firebase Cloud Storage for Firebase is a fast, secure, and scalable object storage service built on Google Cloud infrastructure, designed for app developers to store and serve user-generated content like photos and videos. It offers a powerful, simple, and cost-effective solution, with Firebase SDKs adding Google-level security to file uploads and downloads, regardless of network quality.

- Firebase Hosting

  Firebase Hosting offers fast, secure, production-grade web hosting for developers. It allows quick deployment of web apps to a global CDN with a single command. While optimized for static and single-page apps, it can also work with Cloud Functions or Cloud Run to host dynamic content and microservices.

- ReactJS for Frontend

  React is a JavaScript library for building user interfaces, commonly used for developing single-page applications and mobile applications. It uses a component-based approach, allowing developers to build complex UIs by composing simple and reusable components.

React also uses a virtual DOM for efficient updates and rendering, making it fast and efficient for dynamic applications.

- Smart contract Solidity:

  A contract in the sense of Solidity is a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereum blockchain. We have used the contract to create the necessary functions for the auctions such as add the auctions, end the auctions and place new bids.

- Third Web:

  ThirdWeb API is a robust platform designed for the deployment and interaction of blockchain smart contracts. It simplifies the complex processes involved in managing smart contracts by offering intuitive tools and services. With ThirdWeb, developers can easily deploy, manage, and interact with smart contracts on various blockchain networks. The API supports transaction handling, contract interactions, and offers monitoring tools to ensure seamless integration and operation. By leveraging ThirdWeb API, developers can focus on building decentralized applications (dApps) without worrying about the underlying blockchain intricacies, enabling faster development cycles and more secure contract management. This makes it an essential tool for developers looking to streamline their blockchain projects.

- Sepolia Blockchain:

  Sepolia is a testnet blockchain network designed for the development and testing of smart contracts. It offers a controlled environment where developers can deploy and experiment with their smart contracts without incurring the costs and risks associated with the mainnet. Sepolia mimics the behavior of the main Ethereum network, providing a realistic testing ground for decentralized applications (dApps) and smart contract functionalities. By using Sepolia, developers can identify and fix potential issues, optimize their code, and ensure the reliability and security of their smart contracts before migrating them to the mainnet. This makes Sepolia an invaluable tool for blockchain developers focused on delivering robust and secure applications.

  Distributed Ledger Technology (DLT) is the foundational technology behind blockchain networks. Each participant, or node, maintains a copy of the ledger, ensuring transparency and security. Blockchain, a type of DLT, organizes data into blocks that are linked together in a chronological order. This structure ensures that once data is recorded, it cannot be altered without altering all subsequent blocks, providing a high level of data integrity and security. DLT's decentralized nature eliminates the need for a central authority, reduces the risk of data tampering, and increases trust among participants. It is widely used in various applications, from financial transactions to supply chain management, due to its ability to provide a secure, transparent, and immutable record of information.

4. Requirement Analysis

In the Initial phase of development, we have researched for the development of different Auction websites and their functionalities.

Client side:

- Create an account
- Login

- Authentication
- View the Auctions
- Bid for the Auction
- Ethereum wallet sign-in
- Ethereum wallet sign-out
- Payment
- Logout

Admin side:

- Login
- Authentication
- Ethereum wallet sign-in
- Ethereum wallet sign-out
- Add Auctions
- Remove Auctions
- Payment status

REQUIREMENT ANALYSIS DESCRIPTION

The **Create Account** operation allows a new user or guess who has just had access to the software to be able to register and access the software functions. Registration is required since we need user rating movie data.

The **Login** operation allows a user who already has access to the application to be able to reconnect to the system by entering the same data that he entered during the previous registration period.

The **View and Bid for the Auctions** operation allows the user to select for their desired auction and bid for them by entering the amount they desire to pay for the entity.

The **Payment** operation allows the user to purchase the entity when the bidding time has been completed and the user has finally won the auction.

The **Logout** operation will allow a logged in user to logout from system. Users will need to login again if they need to do any other action on the next connection.

The **Ethereum wallet sign-in** allows the users to connect to the blockchain wallet in order to make the transactions on the blockchain network.

The **Ethereum wallet sign-out** allows the users to disconnect from the blockchain wallet and able to change to different wallet connection if needed.

The **Add the Auction** operation allows the user to upload their entities on the website by selecting the minimum amount for it and also choosing the amount of time to bid for the auction.

The **Remove the Auction** operation allows the user to remove their particular bids that they have uploaded so that it is not available anymore for the users to bid for that particular aucitons.

The **Payment Status** operation allows the user to track the status of their payment upon completion of their payment to the website.

4.1. Functional requirements

Functional requirements refer to the product features or functions that must be implemented by the development team to enable users to carry out their tasks. It is important to make these requirements clear for both the development team and stakeholders. Typically, functional requirements describe the system's behavior in specific circumstances. Below are functional requirements for software:

FR-1: User Authentication Management:

- Register user
- Verify user
- Login User
- Connect to wallet
- Wallet verification

FR-2: Bid Management:

- Choosing the auction
- Bid for auction
- Add to cart
- Update bid amount in wallet contract

FR-3: Payment Management:

- Payment for the bid
- Payment through Blockchain wallet

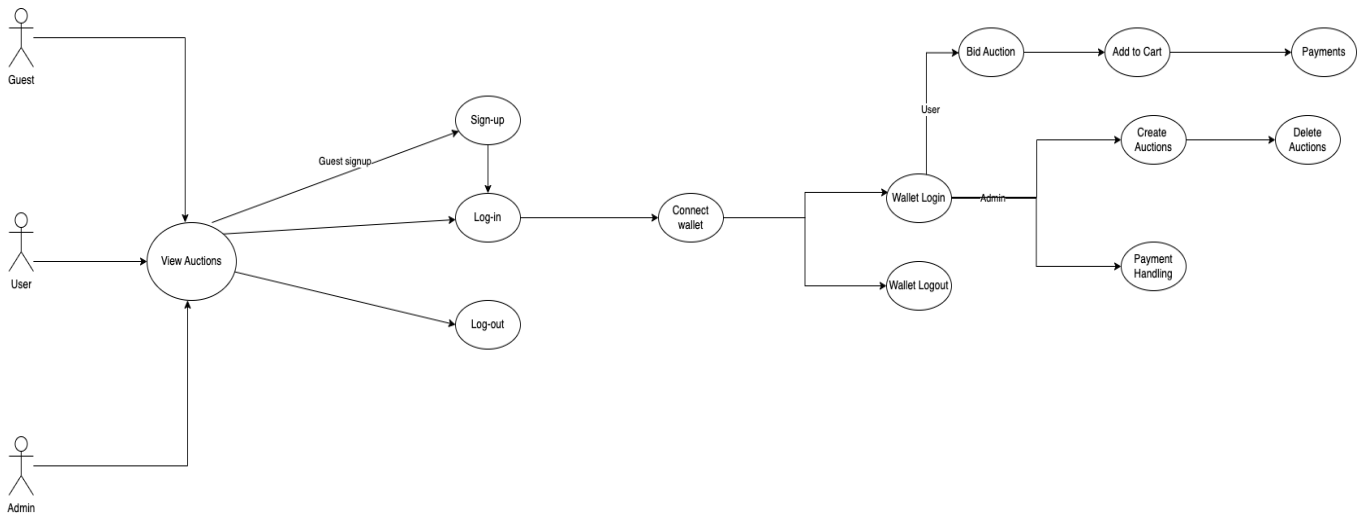
FR-4: Admin Authentication Management:

- Admin Login
- Admin Get user details
- Admin Get Blockchain wallet

FR-5: Admin Auction Management:

- Create Auction
- Admin wallet contract creation
- Admin wallet contract handling
- Delete Auction
- Uploading Auction image

FR-6: Admin Payment Management:



- Payment status
- Get user payment details
- Get payment details from Database
- Get payment details from wallet blockchain

4.2. Non-Functional requirements

Nonfunctional requirements, not related to the system functionality, rather define how the system should perform. Below are non-functional requirements for software:

NFR-1: Friendly Graphic User Interface

NFR-2: Multiplatform compatible

NFR-3: Fast response to the user

NFR-4: Auctions should be available

NFR-5: Multi-user handle

NFR-6: Blockchain availability

4.3. Use case diagram

The specifications for the first software prototype involve a thorough analysis of the customer requirements, which is important and requires careful attention as it determines whether the customer's needs goods are met or not. A "use case diagram" will be used to describe the relationship clearly and accurately between users, admin and the actions they can make while using the software.

5. Backlog Refinement

From the Requirements Analysis, as a Product Owner, I have refined the backlog as follows (priority order is from top to bottom).

auction webpage Team

New Work Item | View as

Backlog | Analytics

Order	ID	Title	Assigned To	Priority	State	Tags
1	1	Auctions		1	To Do	
	2	Research on Auctions website functioning		2	To Do	
	8	Authenitcation (User and Admin)		1	To Do	
	14	Home Page		1	To Do	
	19	Payment handling		3	To Do	
	22	Deployment		2	To Do	
	23	Testing		3	To Do	
	26	Smart contract		2	To Do	
	32	Deployment of Blockchain		1	To Do	
	35	Testing of Blockchain		3	To Do	

6. Project Estimation

There will be Six approaches for building our Auctions website and their corresponding issues take 6 sprints to complete.

At the same time, we can complete the authentication system in 1 Sprint and Development in 1 Sprints.

There will be one last Sprint for combining components into a complete system.

All the designs will be done in Sprint where the feature or system is in “Active” status. The final test phase will take place when all the system is done as a system test.

So, there will be 6 sprints:

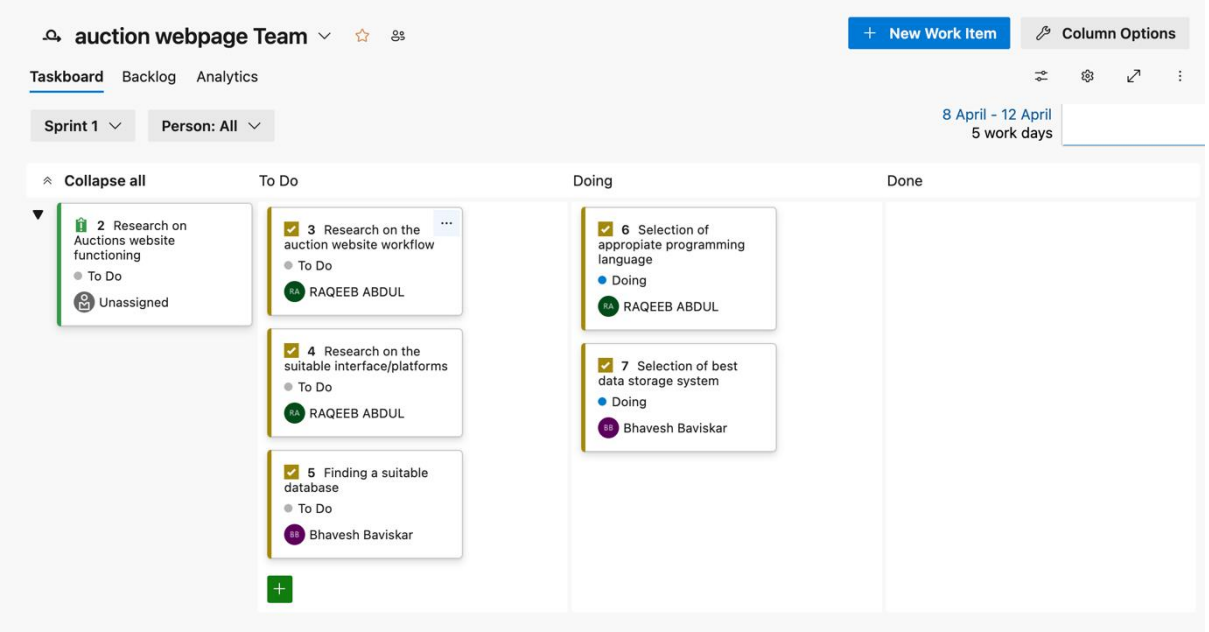
1. Sprint 1 runs from 8th April to 12th April. Issues: 1.
2. Sprint 2 runs from 15th April to 19th April. Issues: 2.
3. Sprint 3 runs from 22nd April to 26th April. Issues: 3.
4. Sprint 4 runs from 6th May to 10th May. Issues: 4.
5. Sprint 5 runs from 20th May to 24th May. Issues: 5.
6. Sprint 6 runs from 27th May to 31st May. Issues: 6.

Sprint Cycles

1. Sprint 1

1.1. Planning

As we discussed in Project estimation, In sprint 1, Issue Research is taken care of and all the tasks of these issues are as below.



1.2. Implementation

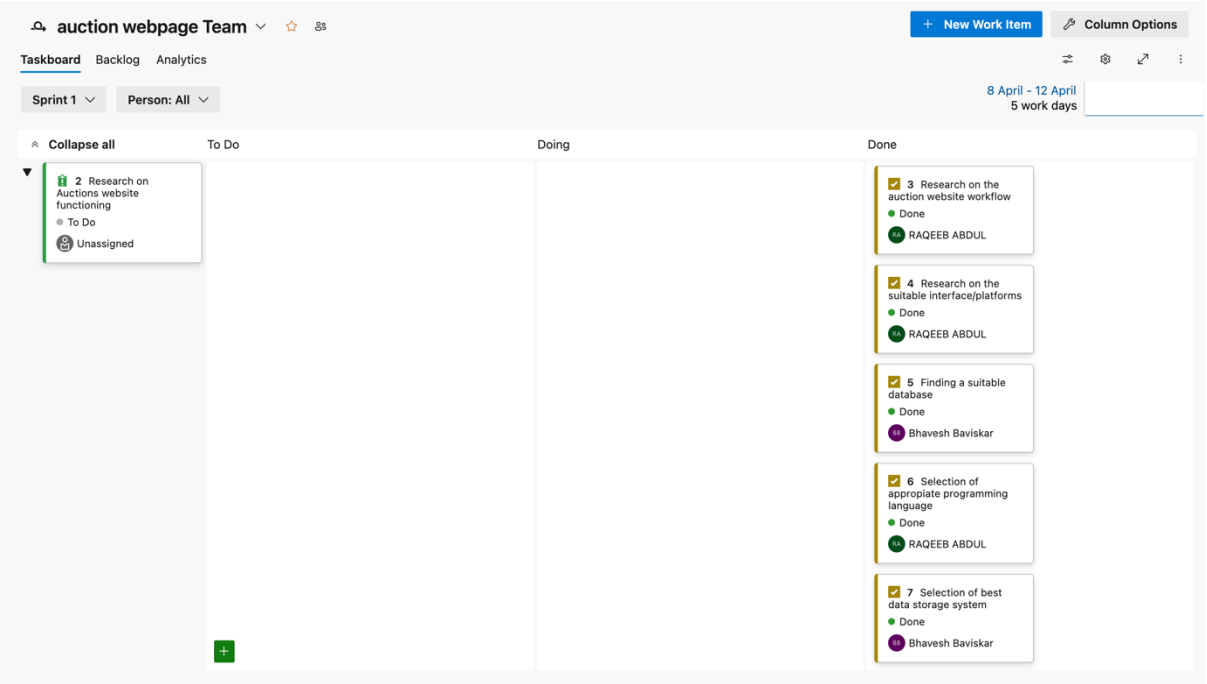
In the First sprint, our primary focus was on the research and selection of the technology stack for the auction website. This foundational work set the stage for future sprints by addressing key technical decisions early on.

- The first step of this Sprint involved thoroughly analyzing the auction website workflow. This was crucial for understanding the core functionalities your platform needed to provide.
- Understanding these components early allowed the team to prioritize key technical choices that align with these needs. This is reflected in your decision to use React, NodeJS, and Firebase, all of which are well-suited for real-time web applications.
- The team has selected React, Node.js, and Bootstrap for frontend, backend, and styling, respectively.
- The team decided to go with Firebase Firestore as the database solution. Firebase is a NoSQL cloud database that offers scalability, real-time syncing, and ease of integration,

making it a strong choice for an auction platform. Its real-time capabilities align perfectly with the dynamic nature of auctions, where data changes (like bids) need to be reflected immediately.

Highly successful in setting the technical foundation for the project. Our team's decision to research the auction workflow and select the right technologies ensures a smooth development process moving forward. By choosing tools that are cloud-native, scalable, and suitable for real-time applications (React, Node.js, Firebase), This sprint ensures that the auction platform will be responsive, secure, and ready for deployment once key features are built. As we move into the next sprints, continue to document your decisions and consider adding automated testing and performance monitoring to ensure your technology stack can handle increased traffic and complex auction scenarios.

1.3. Review



1.3.1. Sprint goals

- Research on the auction website workflow.
- The goals in this sprint focused on fundamental research and technology selection, which are crucial early steps in the project.
- The emphasis on researching workflows and platforms ensures that the choices are informed by a solid understanding of the auction domain.
- Researching databases early on and analyzing their suitability to the application provides a strong technical foundation for the web app, especially considering scalability and future integration needs.
- Analyzing the best suitable database for the web application.

1.3.2. Tasks completed

- Selecting a tech stack (React, Node.js, Firebase) in the early sprint made a decisive progress and lays a strong groundwork for development.
- The decision to go with Firebase Firestore is particularly noteworthy for its real-time data syncing capabilities, which are critical for auction systems where live bidding is key.

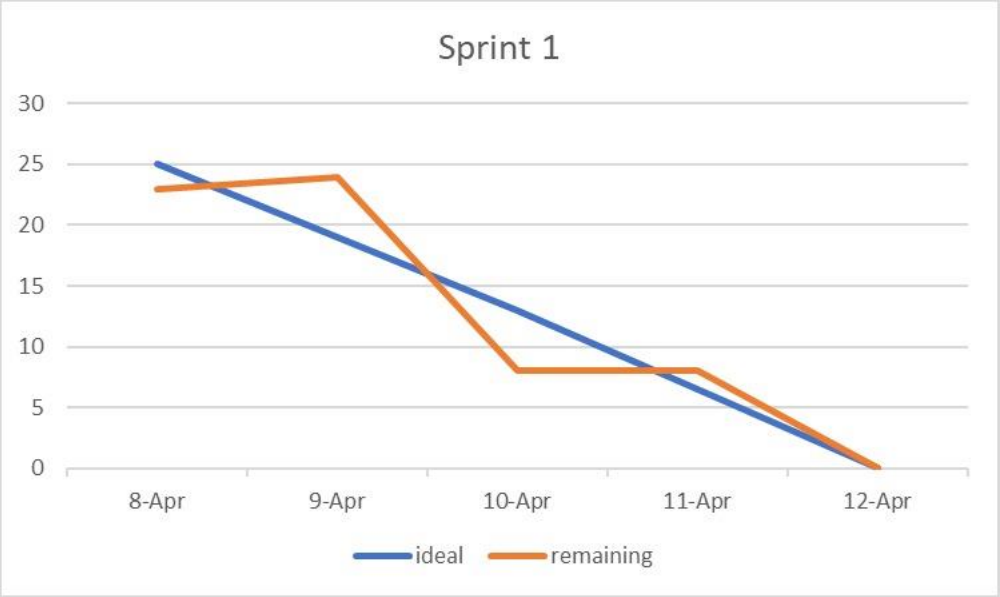
1.3.3. Testing Phase

During this sprint we conducted the testing through manual approach, at the end of the sprint we were able to select Firebase storage for image storing, firebase database for other information, firebase authentications for user authentications and verifications, and selected ReactJs script as a programming language for the frontend and backend.

Our team performed manual testing during this sprint to verify the selected stack and system components. This is a great practice in the early stages of a project. Manual testing allowed us to validate that Firebase is the optimal solution for image storage, database management, and user authentication, reducing future integration risks. The choice to deploy a local test system was prudent, as it allowed to compare database options without overcomplicating the early development process.

1.4. Retrospective

1.4.1. Burndown chart

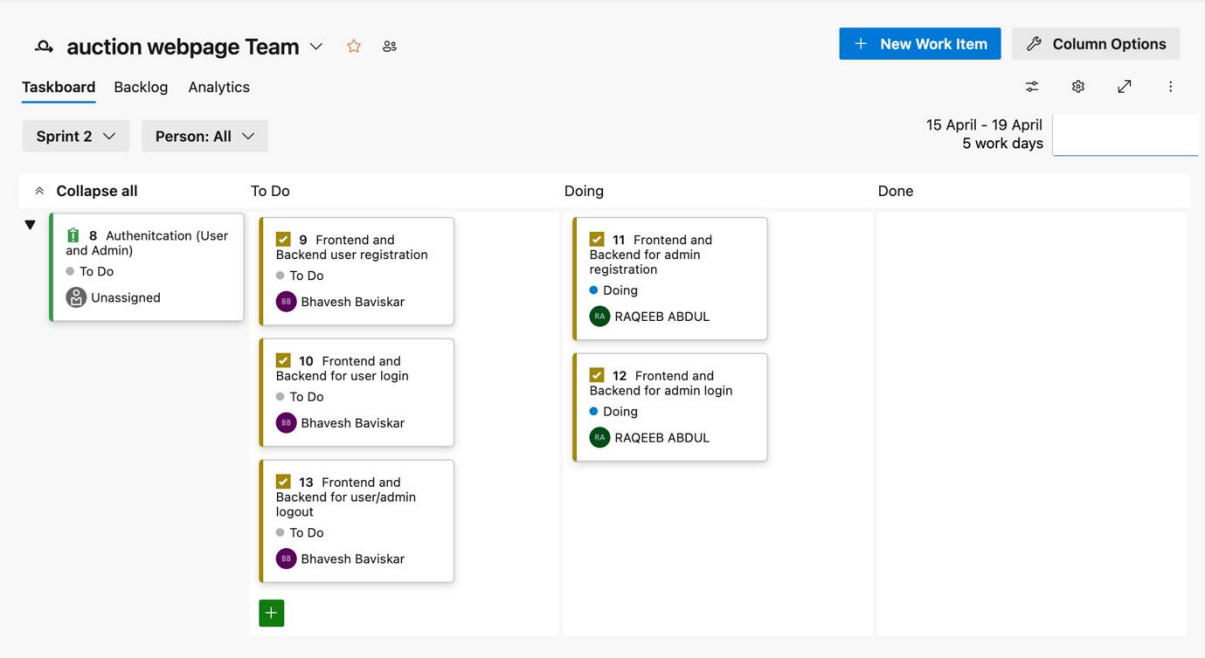


In the first sprint, we mostly focused on the storage system and were trying to analyze which is more suitable. So at the start this was the main issue for the delay and also in the later stages we were able to minimize our delay by doing the tests faster. We were able to coordinate with the team for any changes and information exchanges between the team.

2. Sprint 2

2.1. Planning

As we discussed in Project estimation, In sprint 2, Issues taken care of and all the tasks of these issues are as below.

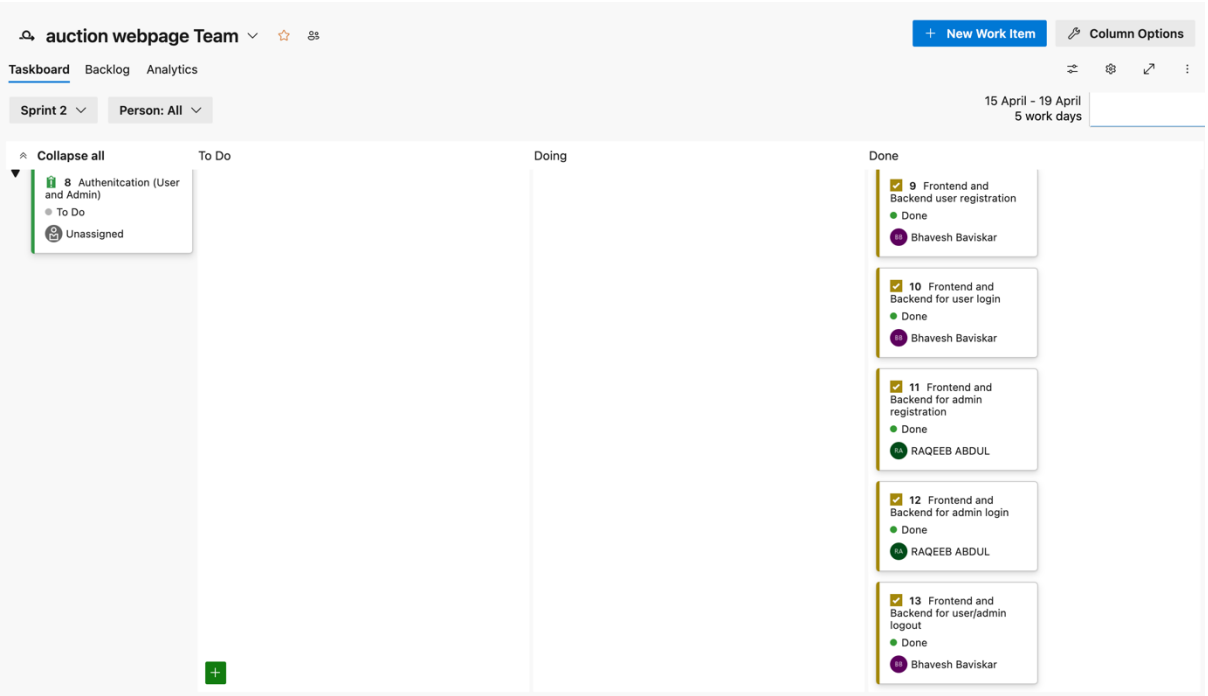


2.2. Implementation

The second sprint of the auction website project focused primarily on user authentication and registration, a critical feature that ensures secure access for both users and admins. This sprint is essential as it lays the groundwork for secure user management and platform integrity.

- This sprint addressed the key requirement of handling authentication for both users and admins, ensuring proper access control to the platform. The system was designed to differentiate between these two types of users, allowing for different permissions and access to platform functionalities.
- The frontend handled the registration, login, and logout flows for both users and admins. Using ReactJS for the frontend made it easier to create dynamic and responsive login forms and registration pages. Firebase Authentication was likely implemented to handle the login system. Firebase Authentication offers ready-to-use UI components and secure methods like email/password authentication, social logins and OAuth integration.
- Nodejs on the backend processed authentication requests, handling login validation and user data securely. It likely communicated with Firebase Authentication to verify credentials and handle session management.
- Handling the Logout functionality for both users and admins was another key task completed in this sprint. The implementation here likely involved in Frontend were clearing the authentication token or session stored in the browser. For example, when a user logs out then the Firebase would terminate the session.
- For Backend, Firebase Authentication automatically handles session expiration and logout, ensuring that users are properly logged out from all devices if needed.

2.3. Review



In this sprint, the main goal was to implement a robust authentication system for both users and admins. By utilizing Firebase Authentication, the team ensured a seamless and secure way for users to register, log in, and log out, while handling the necessary backend operations. This sprint was vital for establishing core security measures and a smooth user experience.

2.3.1. Sprint goals

- The primary focus of this sprint was to achieve the following authentication features on both the frontend and backend.
- User Registration: Implemented using Firebase Authentication, allowing users to create accounts. The Firebase system automatically handled tasks such as email validation, password hashing, and storing user credentials in the cloud.
- User Login: This allowed existing users to authenticate themselves and log in. Firebase provides tools to manage sessions, ensuring that users remain logged in even after refreshing pages or navigating the site.
- Admin Registration and Login were similar to user registration and login but with a differentiation in privileges. The team likely used Firebase roles or custom claims to handle admin authentication and provide them with enhanced capabilities such as managing auctions, verifying users, etc.
- Logout Functionality: Both users and admins had the ability to log out, and Firebase automatically handled session termination. This prevents unauthorized access after logout.

2.3.2. Tasks completed

- Frontend and Backend implementation of Authentication for Admin registration using Firebase services.

- Admin Registration and Login: The team implemented separate authentication flows for admins using Firebase services. Admins were granted elevated access rights compared to regular users, likely through Firebase's custom claims feature. This allows the system to differentiate between admin and user logins and provide appropriate privileges.
- User Registration and Login: Firebase's simple API made it easy to implement user registration and login. Firebase handled common challenges such as preventing duplicate registrations and managing secure password storage. Users could now create accounts, log in, and access the auction platform securely.
- User and Admin Logout: A unified logout functionality was implemented for both users and admins. This cleared the session state in Firebase and the frontend (React), ensuring users and admins were logged out securely and their session tokens invalidated.
- Third-Party Authentication: Though not fully implemented in this sprint, Firebase provides easy integration with third-party login methods, such as Google and Facebook. This feature was briefly tested, offering flexibility for future user logins through these platforms.

2.3.3. Testing Phase

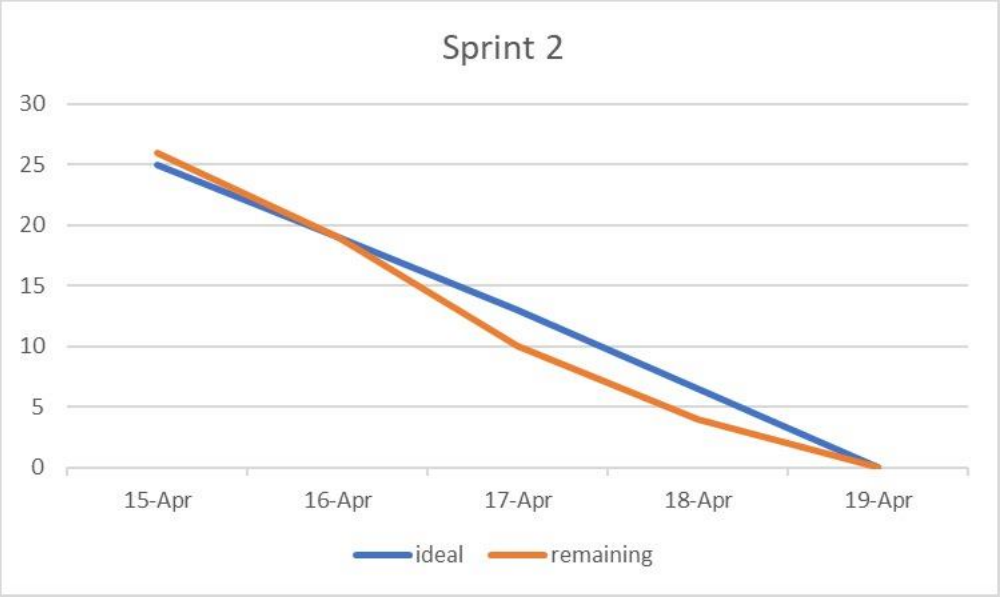
For the sprint, the main focus was in the development of User and Admin login, logout, registration and backend verifications.

- User and Admin Login/Logout: The login and logout processes were tested to ensure they worked as expected. Users were able to register and log in to their accounts without issues. The logout functionality was tested to verify that user sessions were properly terminated.
- Session Management: Special attention was given to session management. The team ensured that once users and admins logged in, they could navigate the platform without needing to re-authenticate. The logout function invalidated the session, preventing access to restricted content after logout.
- Third-Party API Logins: The Firebase Authentication system's ability to support Google and Facebook logins was briefly tested. This was not fully implemented in this sprint but would be considered in future sprints as a convenience feature for users.
- Bugs Identified: Minor bugs related to single login sessions were identified during the manual testing process. These were promptly fixed, and no bug reports were raised in the scrum, indicating quick issue resolution.

Sprint 2 was highly successful in establishing the core authentication features necessary for your auction platform. By leveraging Firebase Authentication, your team efficiently implemented user and admin registration, login, and logout functionality, providing a solid foundation for user management. The combination of real-time session management, strong security features, and flexibility for future third-party logins ensures that the platform is well-prepared to scale as more users join the auction system. This sprint sets a solid technical foundation for the rest of the project.

2.4. Retrospective

2.4.1. Burndown chart

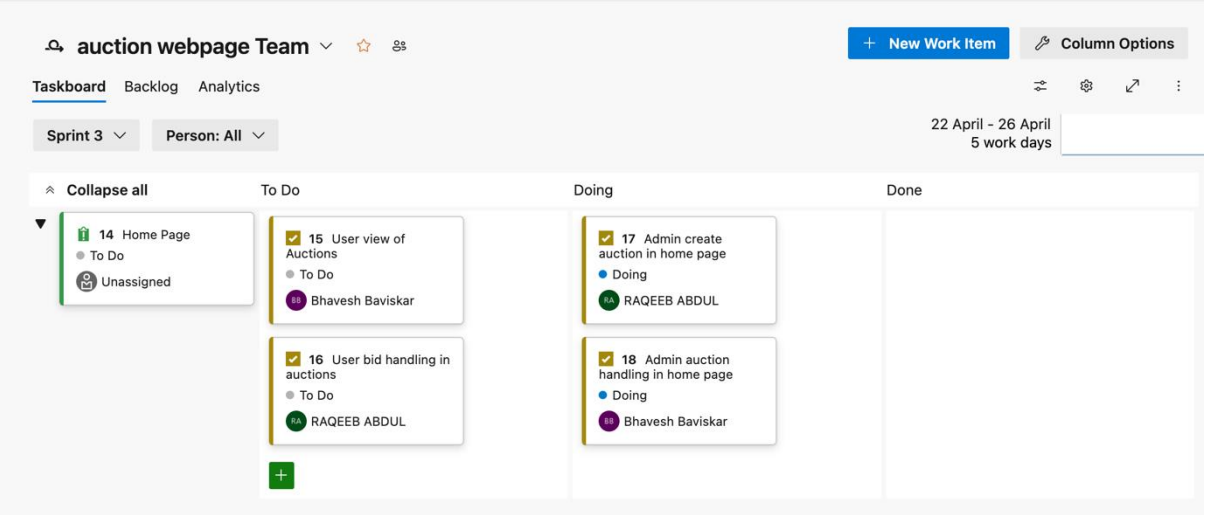


For this sprint as we learned from the previous sprint, we were able to complete the required tasks before the timeline and were able to focus more in the testing parts as this was the main part of the code which should able to handle multiple users as security can be vulnerable in this process.

3. Sprint 3

3.1. Planning

As we discussed in Project estimation, we have mainly focused on the Issue Home page management forUser and Admin by also including tasks that were to be done.



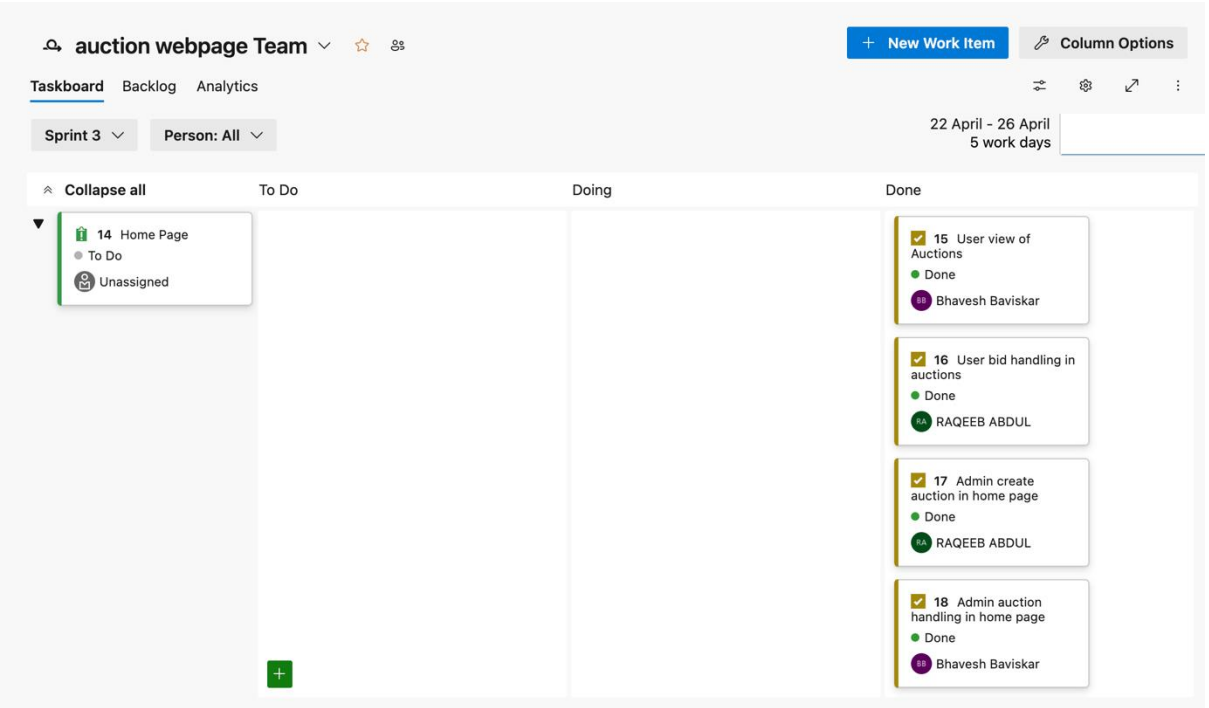
3.2. Implementation

In this sprint, the focus was on building the core auction functionality for both users and admins. Here's a detailed breakdown of the implementation.

- The home page was developed to provide users with an intuitive interface where they could view active auctions. Each auction item would display key details, such as the item name, description, current highest bid, and the time remaining until the auction ends. Auction Listings: The items for auction were pulled from Firebase Firestore, which acted as the main database for storing auction data. This data was fetched and displayed in real-time on the frontend using ReactJS.
- A critical part of this sprint was developing the bidding mechanism. Users could select an auction and place a bid on an item. This functionality was implemented by allowing users to input their bid amount in a form. The bid amount was validated on the client side (ReactJS) to ensure that it met the criteria (e.g., higher than the current bid). Once validated, the bid data was sent to Firebase Firestore, where it was processed and updated for the auction item. Real-Time Updates in Firebase's real-time capabilities were leveraged so that any new bids placed by users would immediately reflect on all other users' interfaces. This ensured a dynamic and competitive auction environment.
- On the admin side, the team built a dedicated interface for auction creation. This involved creating a form where admins could input details such as the item name, description, starting price, auction duration, and other relevant information. Upon submission, the auction data was saved in Firebase Firestore, and the newly created auction was displayed to users on the home page. For Data validation was implemented to ensure that admins couldn't submit incomplete or incorrect auction data (e.g., missing item names or starting prices). The form likely handled errors gracefully, providing feedback to the admin.
- In addition to creating auctions, admins were given the ability to manage existing auctions. This included features such as: Editing Auctions as Admins could modify auction details, such as updating the description or extending the auction duration. Deleting Auctions such If an auction was no longer relevant or valid, admins could delete it from the system. This auction would remove the auction from Firebase Firestore and, as a result, it would no longer be visible on the user side. Authorization and Access Control for Only authorized admins had access to the auction management features. This was likely handled through Firebase Authentication or another role-based system, ensuring that only admins with the necessary privileges could edit or delete auctions.
- Real-Time Bid Updates Every time a user placed a bid, the system checked whether it was higher than the current bid. If it was, the bid was accepted and saved in Firestore. The new bid amount was then pushed to all users viewing that particular auction. Bid Conflict Handling: In case of simultaneous bids, Firebase's real-time database or Firestore handled concurrency, ensuring that only the highest valid bid was recorded.

The team developed crucial features for both user interaction and admin management within the auction system in this sprint. The use of Firebase's real-time capabilities ensured that users had a dynamic experience, while the admin panel allowed for easy auction creation and management.

3.3. Review



After this sprint, significant progress was made on the auction platform, primarily focusing on the admin-side auction creation and management as well as the user-side auction participation. The sprint successfully met its goals by developing both functionalities, ensuring that the website was able to display auctions to users and manage auctions efficiently from the admin side.

3.3.1. Sprint goals

- Admin Auction Creation: Admins were able to create auctions by inputting item details, starting prices, and durations. This was crucial for enabling auctions to go live on the platform.
- Admin Auction Handling: Admins could manage their auctions, including editing or deleting them if necessary. This gave admins full control over the auctions posted on the platform.
- User Side Auction Availability: Once auctions were created, they were made available to users, who could view them on the home page and decide which items to bid on.
- Bid Handling for Users: A functioning bid system was implemented, allowing users to place bids on items, and ensuring that bid changes were reflected in real-time across all users viewing the same auction.

3.3.2. Tasks completed

- Admin Auction Creation: The team successfully developed the feature that enabled admins to create auctions. This was done through a form-based system where admins could enter the required information such as auction title, description, starting bid, and duration. The details were stored in Firebase Firestore and instantly displayed on the user side.
- Admin Auction Management: Admins were able to edit auction details or delete auctions entirely. These actions ensured that admins had control over the entire lifecycle of an auction. This involved secure access control to prevent unauthorized users from making changes.
- User-Side Auction Display: Auctions created by admins were dynamically displayed to users on the home page. The frontend design ensured that users could easily see available items, along with relevant details like the current highest bid and time left in the auction.
- Bid Handling for Users: Users could place bids on auction items, and the bid-handling system was able to process these in real-time. If multiple users placed bids simultaneously, the

system ensured the highest bid was updated accordingly and displayed across all user interfaces.

3.3.3. Testing Phase

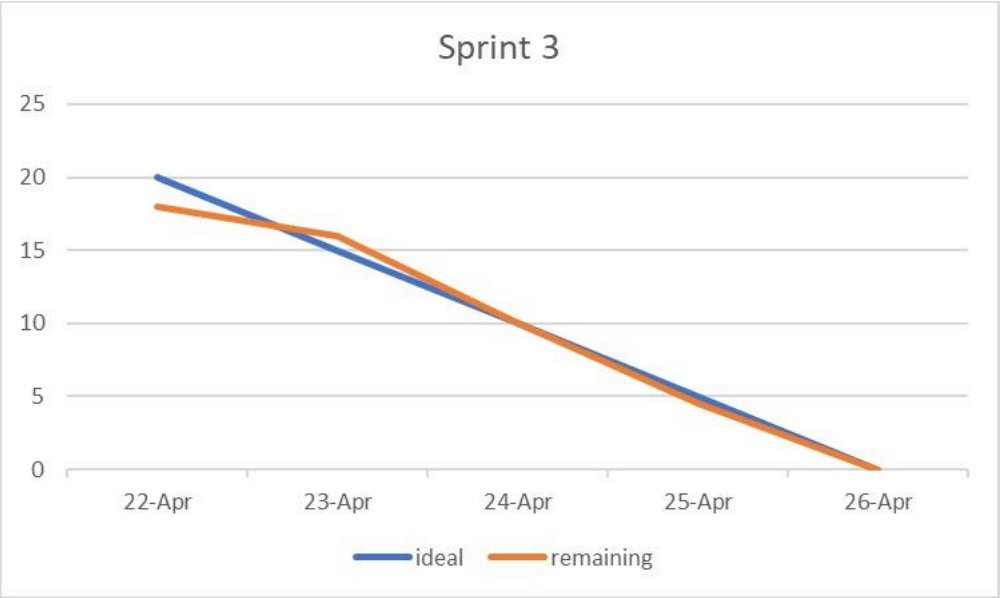
For this sprint, the testing process was thorough, covering both the admin side and the user side to ensure functionality was working as expected.

- Admin Side Testing - Auction Creation: The team tested the creation of multiple auctions to ensure that they were stored in the database correctly. The auctions were checked for their proper retrieval on the user side.
- Admin Side Testing - Auction Handling: The admin features for editing and deleting auctions were tested to ensure that any changes made by the admin were correctly updated in the database and reflected on the user interface.
- User Side Testing - Auction Display: The auctions created by the admin were tested for visibility on the user's home page. It was ensured that the auction data was being fetched from the Firebase database accurately.
- User Side Testing - Bid Functionality: The team tested the user-side bidding mechanism extensively. This involved testing both individual bids as well as simultaneous bids from multiple users. The main challenge here was ensuring that the system handled multiple bids correctly in real time. Initially, some bugs were found when multiple users tried to bid at the same time, but these were successfully resolved by the team working together to ensure data consistency.

By the end of the sprint, the platform was fully functional in terms of creating, managing, and participating in auctions. The admin had complete control over auction creation and management, while users were able to interact with auctions and place bids effectively. The sprint set the foundation for future developments by solidifying the core auction functionalities. This sprint was a key milestone in the development of the platform, ensuring that the major auction workflows were implemented and tested successfully.

3.4. Retrospective

3.4.1. Burndown chart

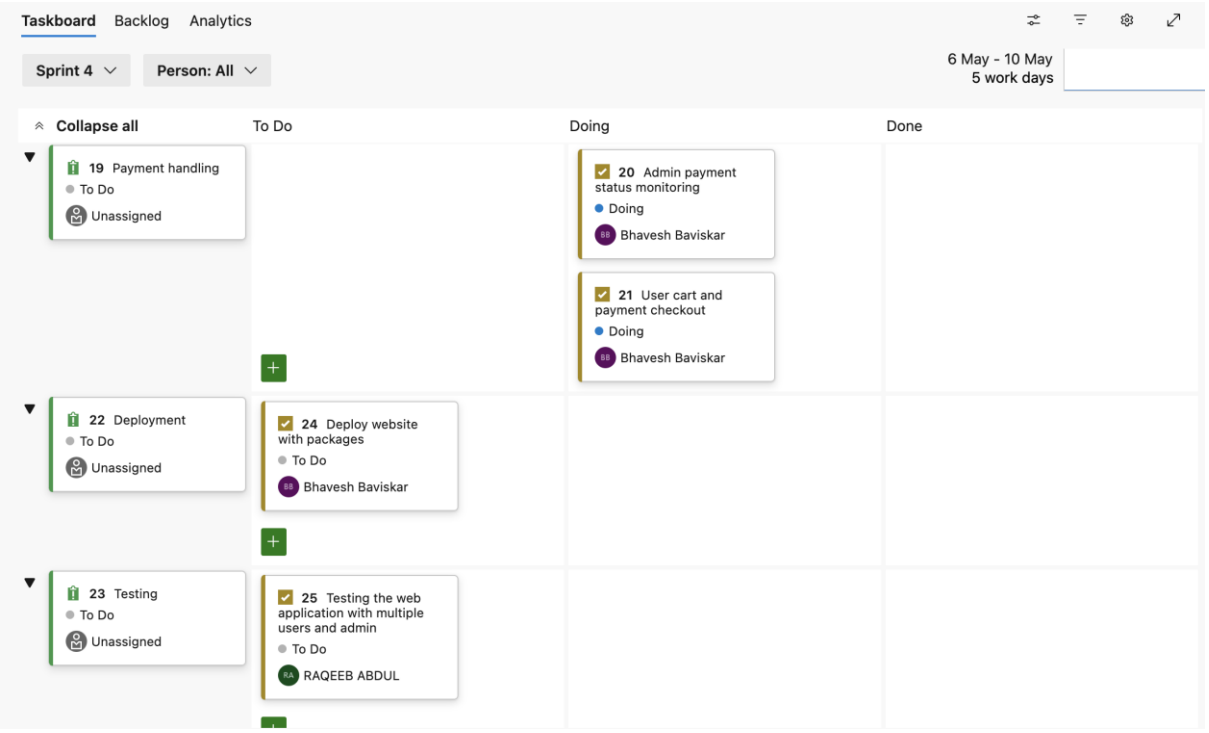


For this sprint, we mostly focused on the adding the auction with the database in the backend. And also for the user bid handling process. There were not many changes in the trend for the burndown as were able to keep the consistent workflow with the tasks in hand.

4. Sprint 4

4.1. Planning

As we discussed in Project estimation, we have focused on the issue Payment handling, Deployment and testing in this sprint which consists of the tasks below.



4.2. Implementation

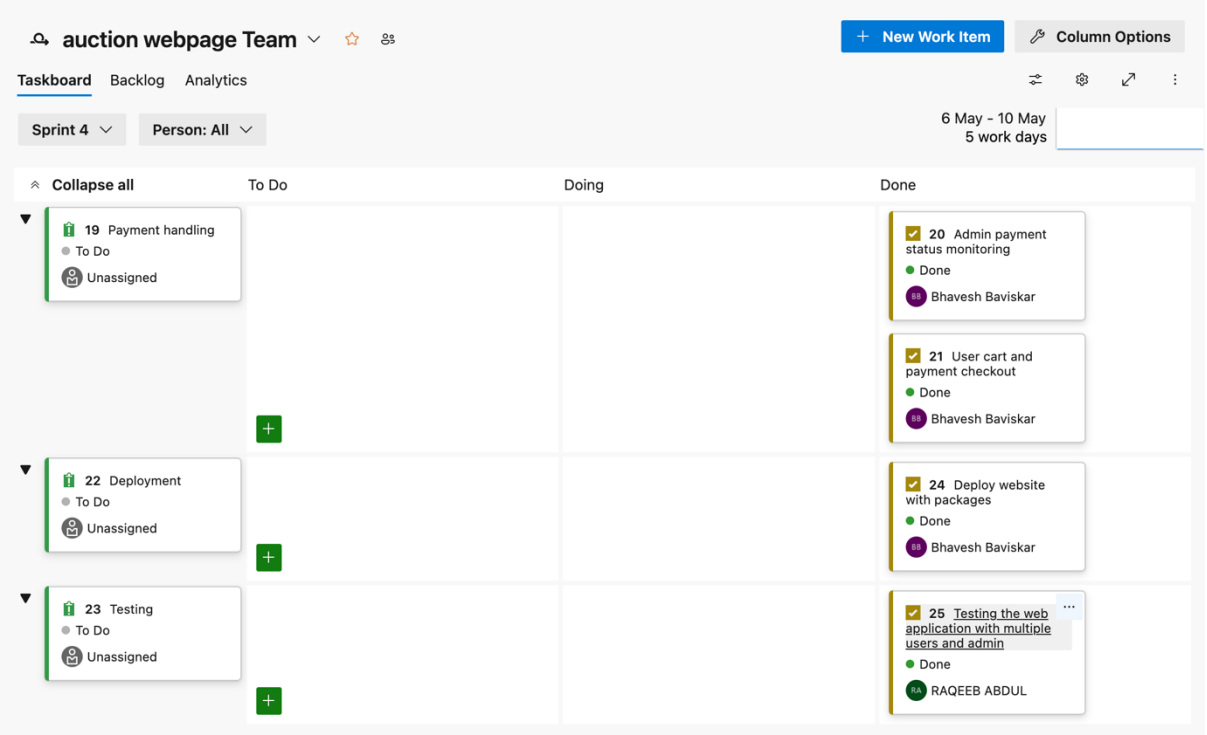
In the fourth sprint, we had completed on Admin side auction handling and manipulation and were still working on the user side auction workflow.

- The admin side functionalities were finalized, allowing admins to manage and manipulate auctions effectively. This included capabilities such as auction creation, modification, and monitoring of auction status. Admins could review bids, confirm auction completions, and facilitate item transfers from sellers to winners.
- Continued development on the user side to enhance the auction workflow. This involved ensuring users could participate in auctions smoothly, place bids, and track their bid status.

Efforts were made to refine the user interface and experience, addressing any issues related to bid placement, auction tracking, and overall navigation.

- Introduced functionality to handle payments for winning bids. This required integrating a payment gateway or system to facilitate transactions securely. Users needed to complete payments upon winning an auction, ensuring that the payment process was smooth and reliable.
- Developed a monitoring system on the admin side to track payment statuses. Admins could view whether the payments had been successfully processed and reached the sellers. This involved implementing notifications or dashboards to provide real-time updates on payment status.
- Deployed the website with all the necessary packages and configurations from Firebase Firestore. This ensured that the web application was live and functional with all features integrated. The deployment process included setting up Firebase Hosting for fast and secure content delivery, ensuring that the application was accessible to users.
- Conducted comprehensive testing with multiple users and admin accounts to verify the functionality of the web application. This testing phase aimed to identify and resolve any bugs or issues, ensuring that the application performed as expected under various scenarios and with different user roles.

4.3. Review



This sprint focused on completing the payment checkout functionality for the user side, finalizing the deployment, and initiating the testing phase of the auction website. The goals included monitoring payment statuses from the admin perspective and ensuring robust testing across various user interactions.

4.3.1. Sprint goals

- Monitoring of the Payment status for Admin using React and Firebase services.
- Monitoring Payment Status: Objective was to Enable the admin to effectively monitor the status of auctions and payments using React and Firebase services. This involved creating a dashboard or interface where admins could view real-time updates on payment statuses and auction progress.
- Testing Web Application: Objective was Evaluate the functionality of the web application from both user and admin perspectives, focusing on payment handling, auction management, and overall system performance. This included testing the system's response to different bids, auction scenarios, and user interactions.

4.3.2. Tasks completed

- Completed Admin Payment Handling: Implementing a Developed and integrated admin interface to monitor payment statuses for auctions. Admins could view details such as auction status, payment completion, and user information related to the auctions. Utilized React and Firebase services to create a dynamic and real-time monitoring system, enhancing the admin's ability to manage and oversee the platform effectively.
- Website Deployment was Process to a Deployed the website using Firebase App Hosting. Initially, the raw code for the React application was deployed, but this was later updated to the build file for optimized Firebase deployment. Ensured that all functionalities of the auction website were integrated and operational in the live environment.

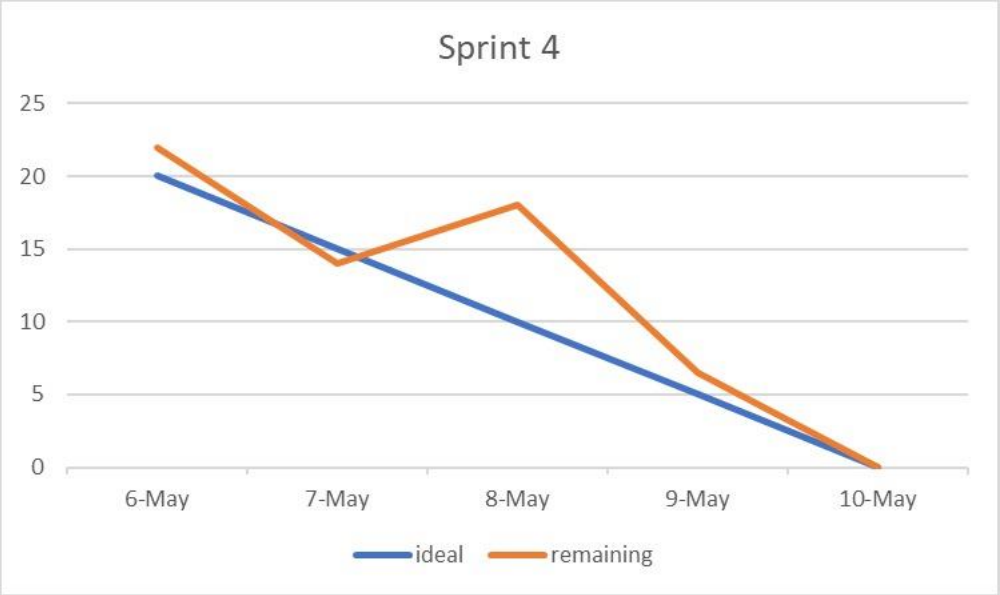
4.3.3. Testing Phase

- User-Side Testing: Process: Tested the payment handling system from the user side by simulating user interactions such as adding items to the cart, placing bids, and completing payment forms. Verified that the database accurately updated user details and bid amounts post-auction. Ensured that payment details were correctly processed and recorded.
- Admin-Side Testing: Process: Conducted tests to confirm that, once a payment was completed, the admin could view user details and item information associated with the payment. This included verifying the accuracy of payment status information and ensuring proper visibility only for admins.
- Deployment Testing: Process: Tested the deployed website hosted on Firebase to ensure that all functionalities, including auction processes and payment handling, were working as intended. Performed security tests, including brute-force attack simulations, to identify and address potential vulnerabilities in the system.

This sprint has successfully addressed key aspects of the auction website, including the completion of payment handling for users, the deployment of the site using Firebase, and comprehensive testing of the application. The achievements in this sprint, particularly in payment monitoring and testing, significantly contributed to the platform's reliability and readiness for further use. The focus on security and functionality ensured that both user and admin experiences were optimized and secure.

4.4. Retrospective

4.4.1. Burndown chart

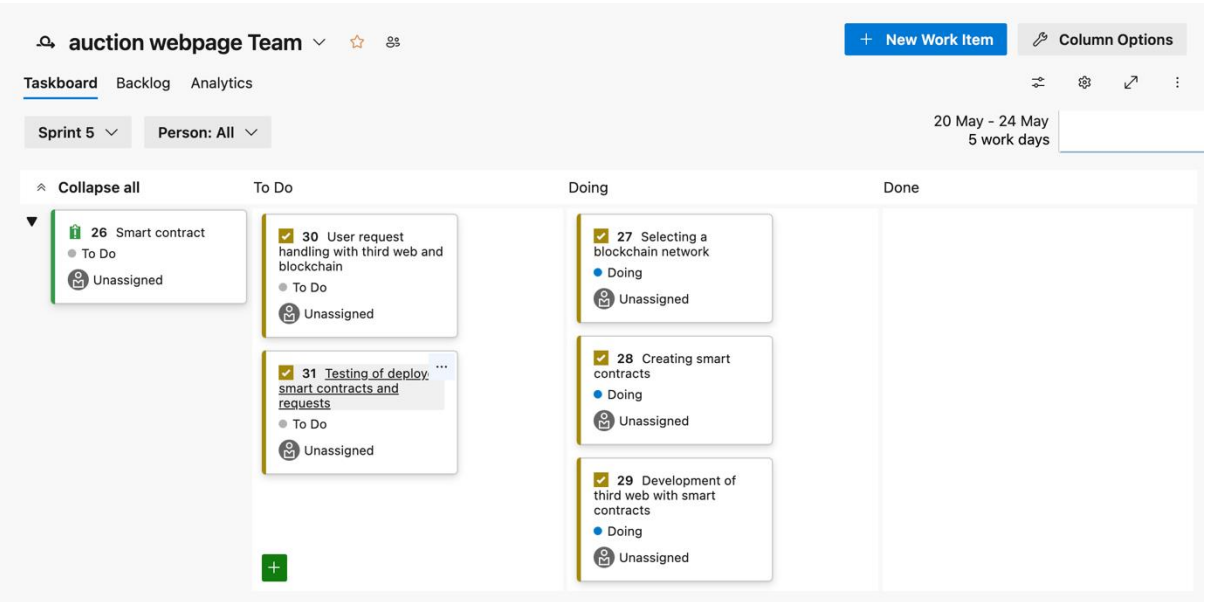


By the end of this sprint, we had completed the payment functionalities and the deployment. Though we started pretty good with work but we faced several issues with the process of the deployment so we had to analyze and redeploy it several times for the testing and checking if all the functionalities are up to the mark. Once the deployment was done, we were able to get on the track and check any further issues after the final deployment.

5. Sprint 5

5.1. Planning

As we discussed in Project estimation, In the sprint 5 we have worked with the issues Smart contracts which consists of the following tasks.

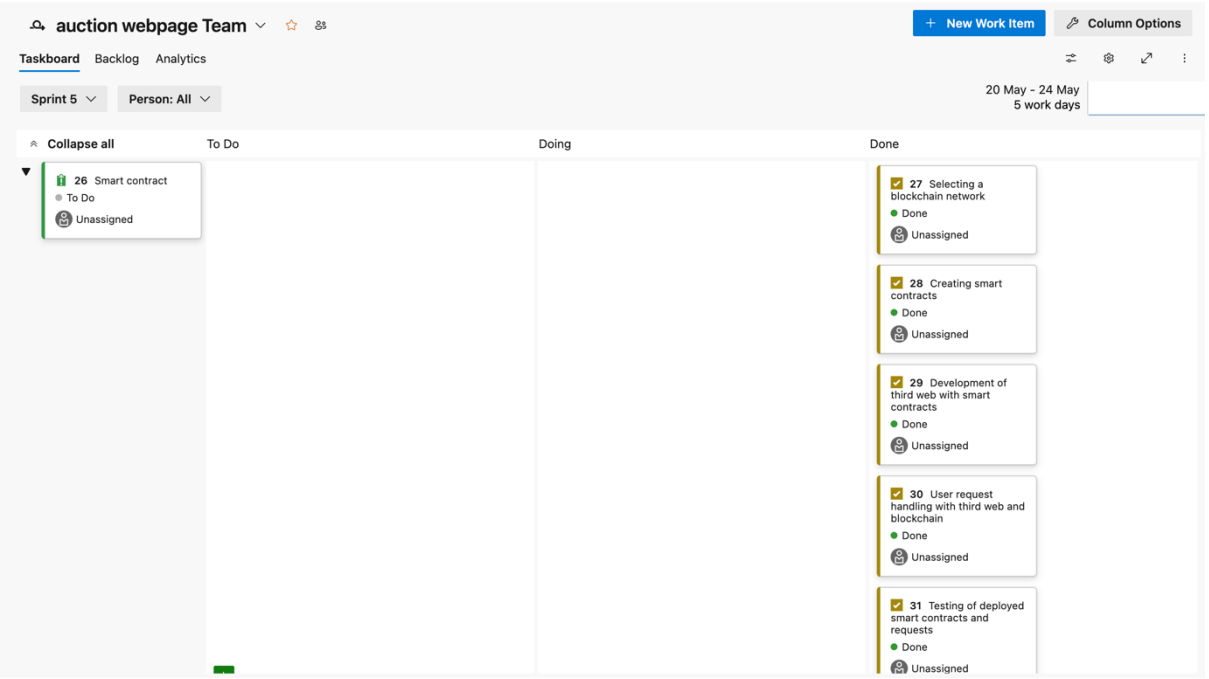


5.2. Implementation

In the fifth sprint, the primary focus was on integrating smart contracts into the auction website using Third Web. This involved selecting, creating, and deploying smart contracts to enhance the platform's functionality with blockchain-based transactions.

- Selection of Smart Contracts: Objective to Choose suitable smart contract templates or frameworks from Third Web that align with the auction website's requirements. Process Research Identified potential smart contract solutions provided by Third Web that fit the auction use case. Selection Chose smart contract templates or created custom contracts that supported auction operations such as bidding, winning, and payment processing.
- Creation of Smart Contracts: Objective to Develop smart contracts tailored to the auction platform's specific needs. Process Contract Design, Defined the logic and rules for the smart contracts, including functions for placing bids, closing auctions, and handling payments. Development Programmed the smart contracts using Solidity (or another appropriate language) to ensure they executed the desired functionalities. Testing is Conducted unit tests and simulations to verify that the smart contracts behaved as expected in various scenarios.
- Deployment with Third Web: Objective to Deploy the created smart contracts to the Ethereum blockchain using Third Web. Process Configuration, Set up Third Web's tools and services to interact with the Ethereum network. Deployment: Deployed the smart contracts to the Ethereum blockchain, making them available for interaction through the website. Integration, Updated the web application to connect with the deployed smart contracts. This involved integrating Third Web's SDK or APIs to enable the website to interact with the blockchain and execute smart contract functions.
- Integration into the Web Application: Objective was to Ensure seamless interaction between the website's frontend and the deployed smart contracts. Process in Frontend Integration to have Modified the web application's frontend to interface with the smart contracts, allowing users to place bids, view auction statuses, and complete transactions directly through the website. Testing Performed end-to-end tests to ensure that the integration worked correctly, including verifying that smart contract interactions were properly reflected in the user interface.

5.3. Review



In this sprint, the team focused on completing the implementation of smart contracts for transaction handling via blockchain networks. This involved selecting a suitable blockchain network, creating and deploying smart contracts, handling user requests, and performing testing.

5.3.1. Sprint goals

- The Blockchain Network: Objective to select the most suitable blockchain network for the development and deployment of smart contracts.
- Creating and Developing Smart Contracts: Objective to Develop smart contracts and integrate them with Third Web to support the auction platform's functionality.
- User Request Handling: Objective to Implement functionality to handle user requests related to bidding and auction management via Third Web and blockchain.
- Testing the Smart Contracts: Objective to Ensure that the deployed smart contracts function correctly and integrate seamlessly with the web application.

5.3.2. Tasks completed

- Blockchain Network Selection: Process a Evaluated various blockchain networks to determine the most suitable option for the project's needs. Considered factors such as network performance, compatibility with Third Web, and integration capabilities.
- Smart Contract Creation and Development: Process to Create the smart contracts necessary for managing auctions and handling transactions on the blockchain. Developed these contracts using Third Web to ensure smooth integration with the web application.
- User Request Handling: Process to Implement features to handle user interactions with the smart contracts. This included handling bid placements and auction management requests via Third Web and blockchain interactions.

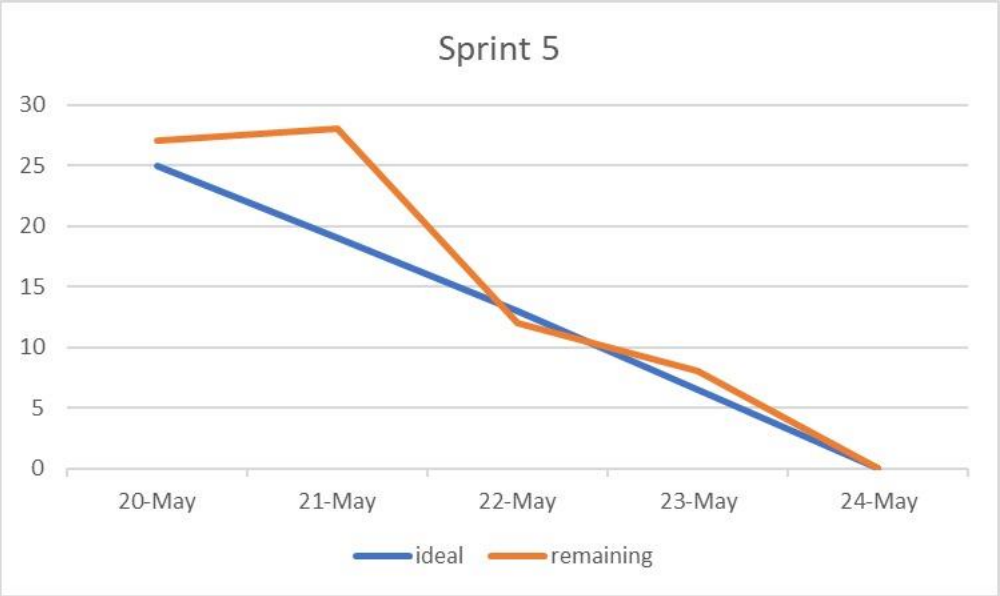
5.3.3. Testing Phase

- Testing Preparation - Network Evaluation: Assessed different Ethereum networks for their suitability for deploying and testing smart contracts. Chose the Sepolia test network for initial deployment and testing due to its compatibility and performance benefits.
- Testing Preparation - Wallet Integration: Used Ethereum mainnet for real transactions and wallet interactions, ensuring that the smart contracts could interact with real Ethereum wallets.
- Testing Process - Deployment: Deployed the smart contracts to the Sepolia test network to test their functionality in a controlled environment before moving to the mainnet.
- Testing Process - Integration: Integrated the deployed smart contracts with the React-based web application. Ensured that only admins could create and manage smart contracts, and users could place bids only when connected to their Ethereum wallets.
- Testing Process - Functionality Testing: Tested the end-to-end functionality of the smart contracts within the web application, verifying that all processes related to bidding, auction management, and transaction handling worked as intended.

This Sprint achieved key milestones in implementing blockchain-based smart contracts for the auction platform. The team successfully selected an appropriate blockchain network, developed and deployed smart contracts using Third Web, and handled user requests effectively. The testing phase validated the functionality of the smart contracts and their integration with the web application, ensuring a robust and reliable auction platform.

5.4. Retrospective

5.4.1. Burndown chart

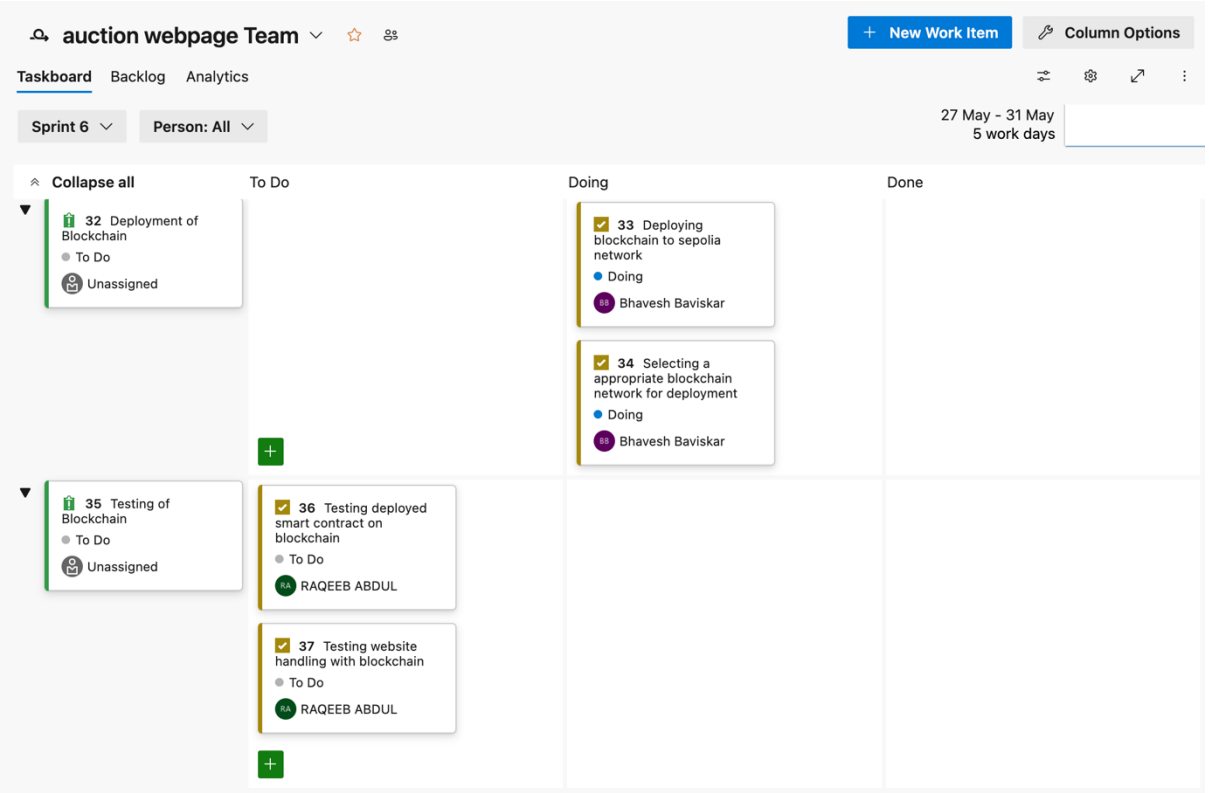


At the end of this sprint, we did not have a very good start for this sprint cycle as we didn't had much knowledge in the area of blockchain and smart contracts of how to interact with them once they are deployed. So we have to use most of the time in learning about them. Once we were able to successfully deploy of smart contract and interact with our website, the further process was easy with adding the different functions in the code of the React file with different conditions and situations with already completed website.

6. Sprint 6

6.1. Planning

As we discussed in Project estimation, In the sprint 6 we had dealt with the issues Deployment and Testing of Blockchain which consists of the following tasks.



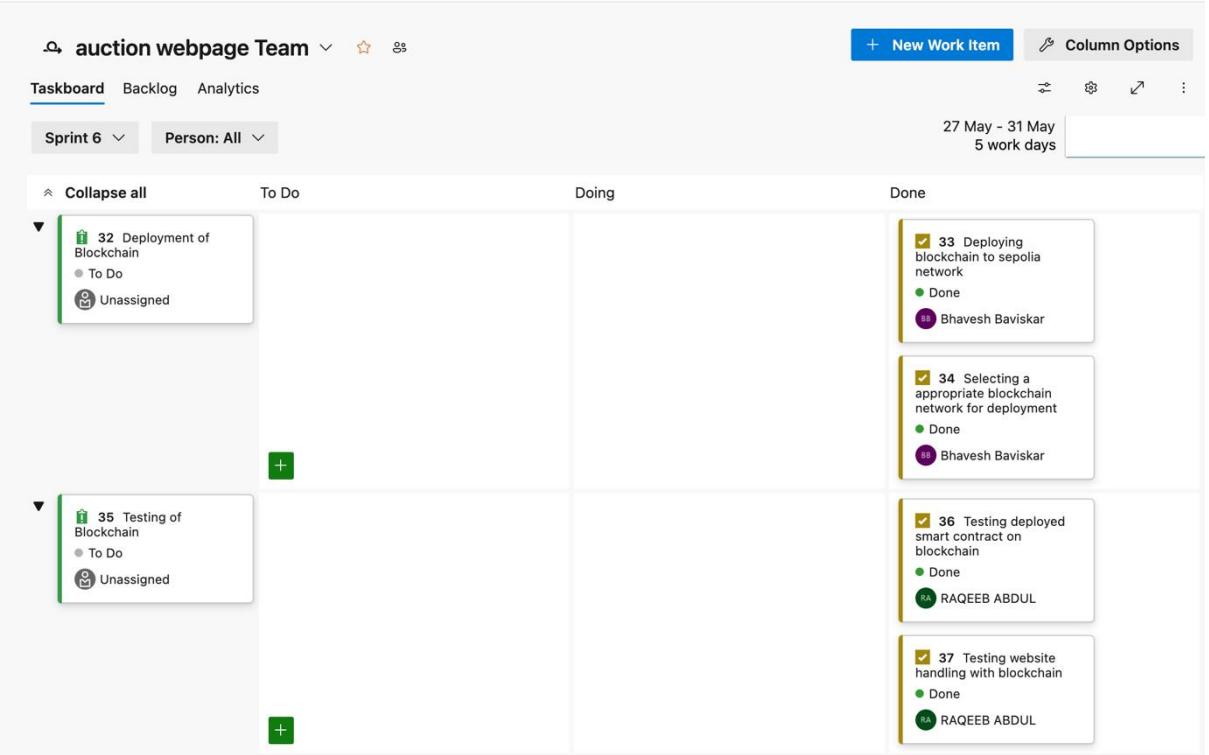
6.2. Implementation

In this sprint, we concentrated on deploying blockchain technologies and smart contracts using the Sepolia test network. This sprint also included the deployment of the application with all associated packages from Firebase Firestore and comprehensive testing of the deployed blockchain components.

- Deployment of Blockchain Technologies - Objective: Deploy the blockchain-based smart contracts and related technologies to the Sepolia test network, ensuring that they are integrated with the web application.
- Deployment of Blockchain Technologies - Process: Finalization of Smart Contracts: Ensured that smart contracts were finalized and thoroughly tested in the development environment before deployment. Configuration: Configured deployment settings to interact with the Sepolia test network, setting up the necessary parameters for smooth deployment.
- Deployment of Blockchain Technologies – Deployment: Utilized Third Web's tools and services for deploying smart contracts to the Sepolia network. This involved interacting with the blockchain through Third Web's interface or APIs.
- Verification: Verified that smart contracts were successfully deployed and accessible on the Sepolia network.

- Integration with Firebase Firestore - Objective: Deploy the web application with all associated packages and configurations from Firebase Firestore to ensure full functionality and data integration.
- Process - Firebase Configuration Setup: Configured Firebase Firestore to handle data storage and management for the auction platform. This included setting up Firestore databases and integrating them with the smart contract functionalities.
- Process - Deployment: Deployed the web application to Firebase Hosting, including all relevant packages and configurations necessary for the application's operation.
- Testing of Deployed Blockchain Components - Objective: Test the deployed smart contracts and blockchain integrations to ensure they function correctly in the live environment.

6.3. Review



In this sprint, we focused on finalizing the deployment and testing of blockchain networks and smart contracts. The sprint aimed to ensure that the blockchain components were successfully integrated and operational within the web application.

6.3.1. Sprint goals

- Selecting an Appropriate Blockchain Network - Objective: Ensure that the selected blockchain network was suitable for deployment and integration with the web application.
- Testing Website Handling with Blockchain - Objective: Verify that the web application effectively interacted with the deployed blockchain components and that the overall system functioned correctly.

6.3.2. Tasks completed

- Deployment of Blockchain with Sepolia Network - Process: Blockchain deployed the smart contracts to the Sepolia test network using Third Web. This involved setting up the smart contracts on Sepolia and ensuring they were accessible and functional.
- Web Application Update: Updated the web application on Firebase Hosting with the latest build file. This ensured that the deployed application reflected the recent changes and integrations.
- Testing of Deployed Smart Contracts- Process: Manual Testing: Conducted thorough manual testing of the deployed smart contracts. This included interacting with the smart contracts through the web application to verify their functionality.
- Multi-User Testing: Collaborated with multiple colleagues to test the smart contracts under various user scenarios. This helped ensure that the system handled multiple users effectively and that transactions were processed correctly.

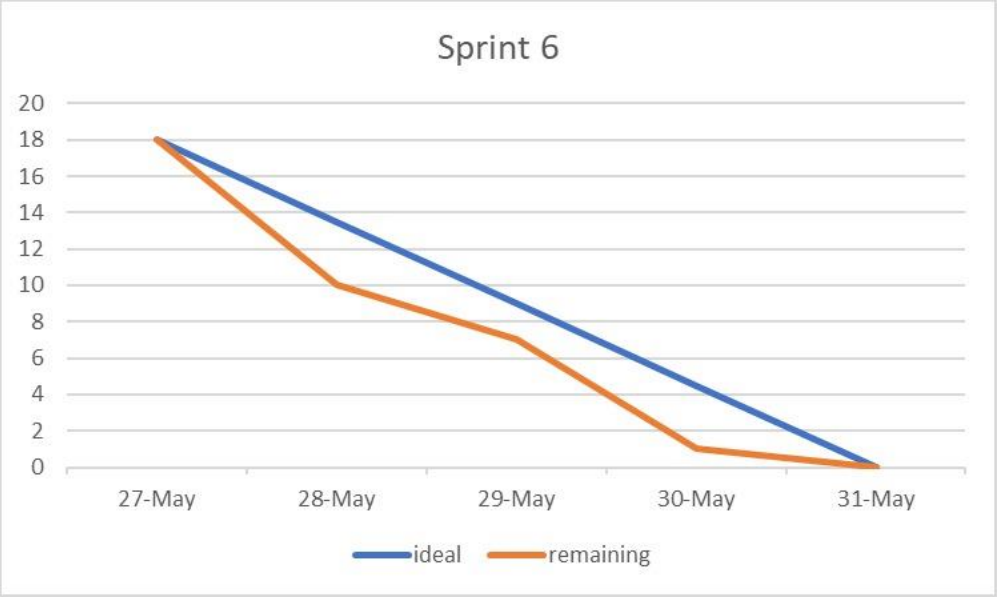
6.3.3. Testing Phase

- Deployment Testing - Firebase Hosting: Updated the deployed application on Firebase Hosting to include the latest build file, ensuring that all recent changes were live. Verified that the web application continued to operate smoothly after the update.
- Deployment Testing - Smart Contract Deployment: Deployed smart contracts to the Sepolia network using Third Web's deployment tools. Ensured that the contracts were correctly deployed and accessible on the blockchain.
- Integrity Testing - Blockchain Integrity: Tested the integrity of the smart contracts and the transactions they handled. This involved checking that the smart contracts performed as expected, and that all transactions were accurately processed and recorded on the blockchain. Ensured that there were no issues with contract execution or data integrity.
- Multi-User Testing - Process: Conducted testing with multiple users to simulate real-world usage and ensure that the system could handle various scenarios. Verified that users could interact with the blockchain components correctly, including placing bids, managing auctions, and completing transactions.

This Sprint successfully achieved the deployment and testing goals for blockchain integration. The deployment of smart contracts to the Sepolia network was completed, and the web application was updated on Firebase Hosting. Thorough testing confirmed that the smart contracts and blockchain interactions were functioning as expected, with no significant issues identified. The sprint demonstrated effective integration of blockchain technology into the auction platform, validating the system's performance and reliability.

6.4. Retrospective

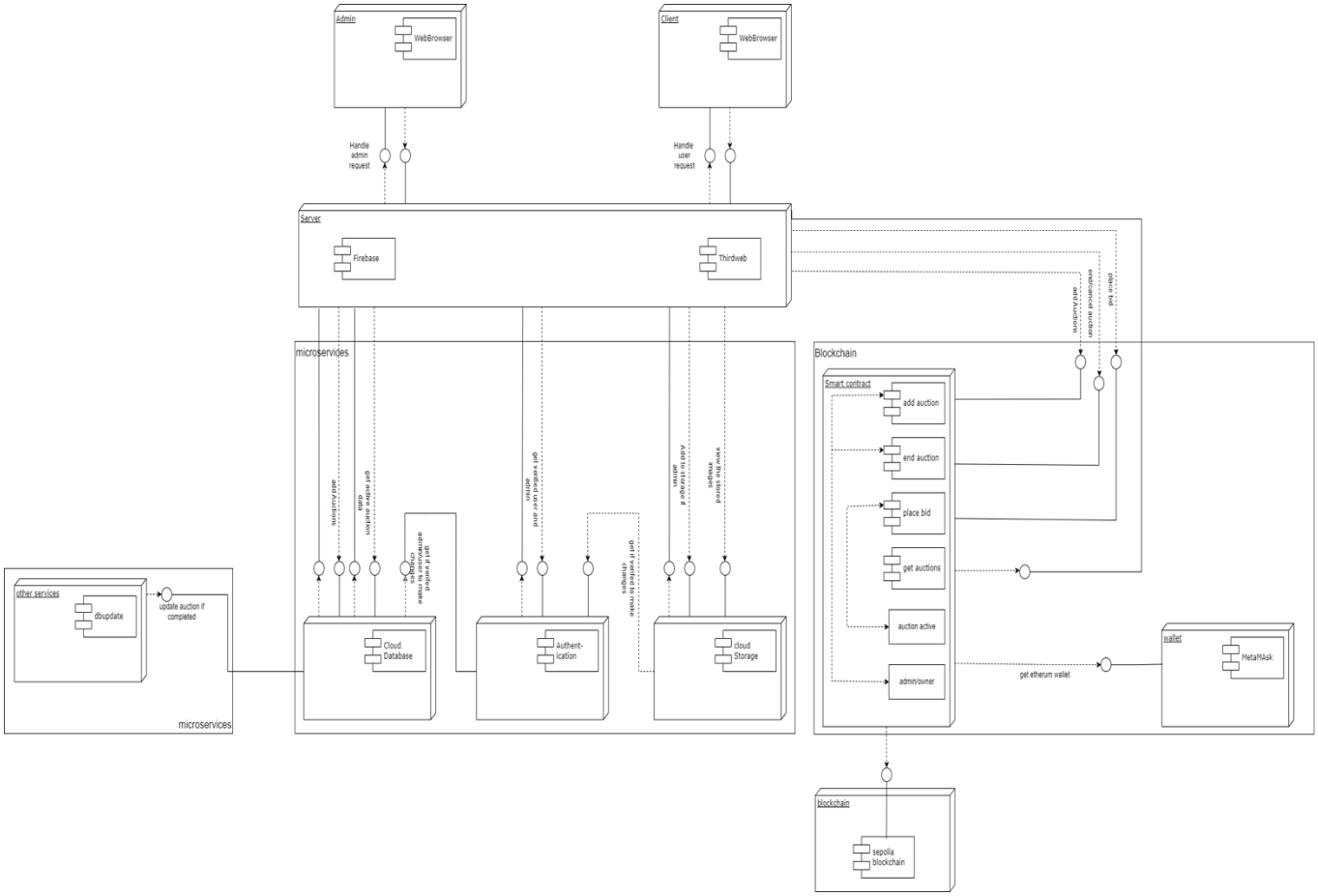
6.4.1. Burndown chart



After this sprint, we were previously knowledgeable for hosting a website with firebase we were easily able to deploy the website which that had been changed with less downtime. There were some issues with deploying the blockchain globally but not very severe as we already tested the deployment with Sepolia blockchain test network. So in this sprint cycle we were able to complete our scheduled work before the end of the cycle.

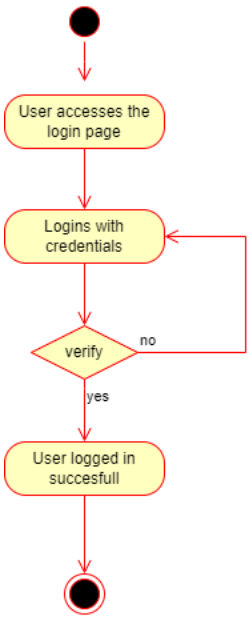
1. Architectural UML design

1.1. Components diagram

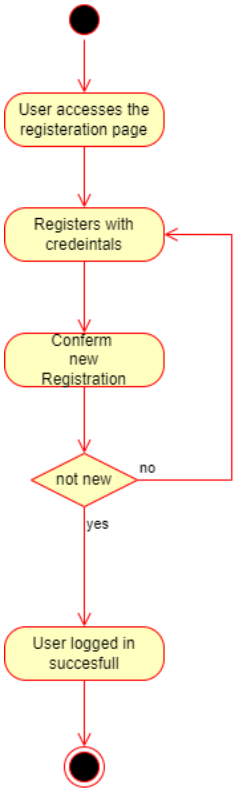


1.2. Activity design

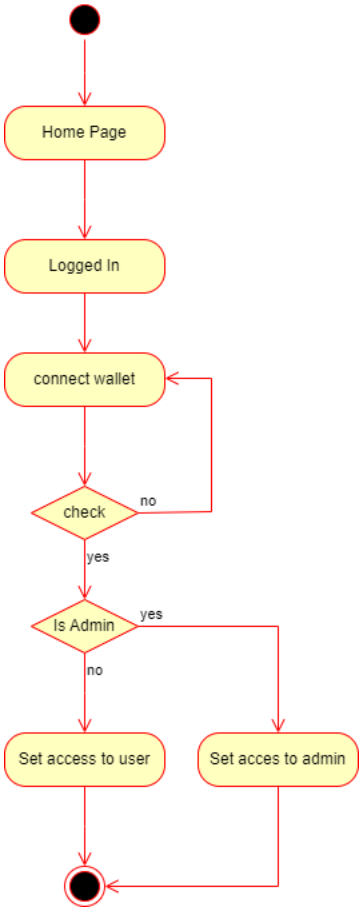
1.2.1. AD user Login:



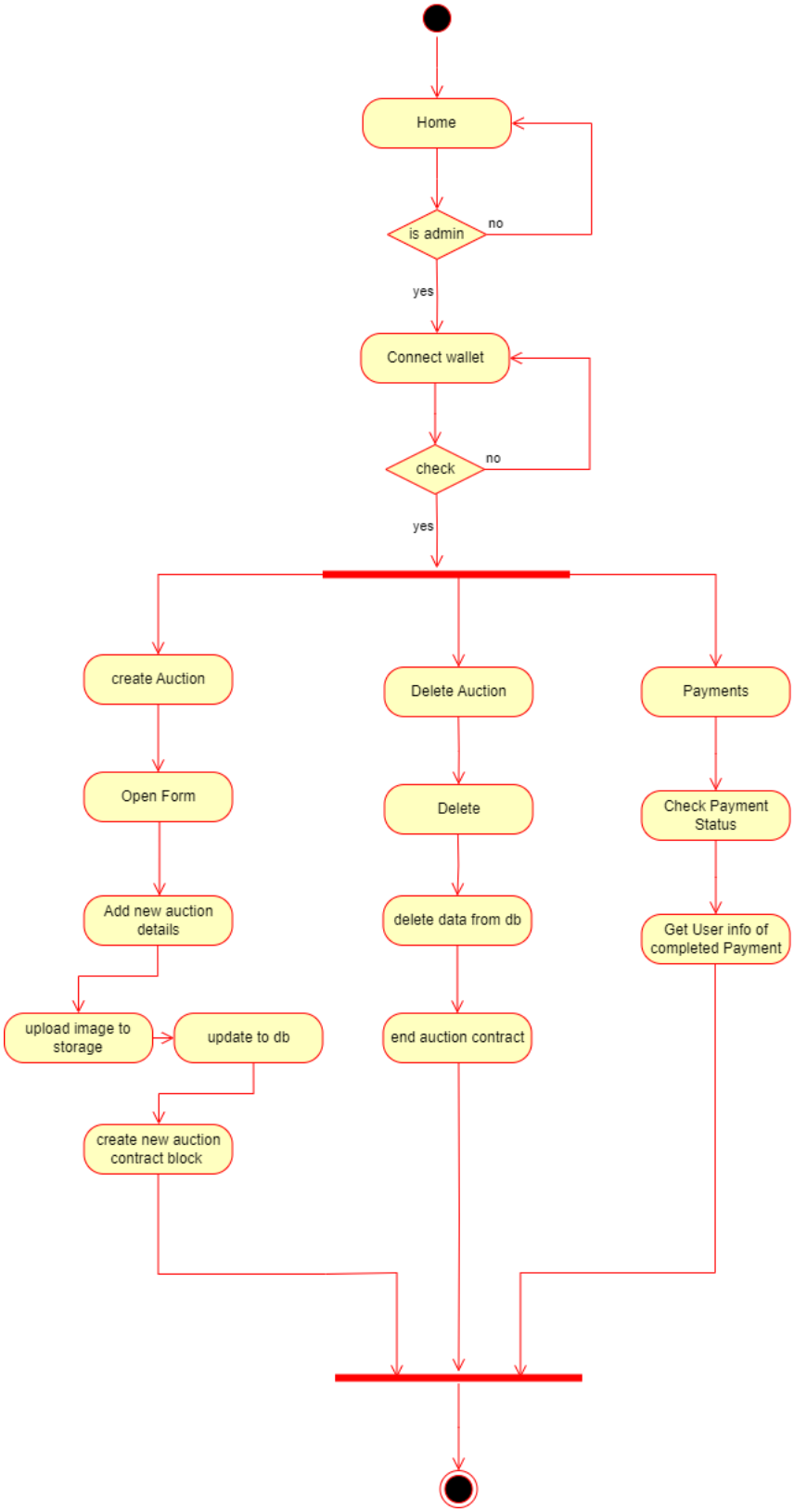
1.2.2. AD user Signup:



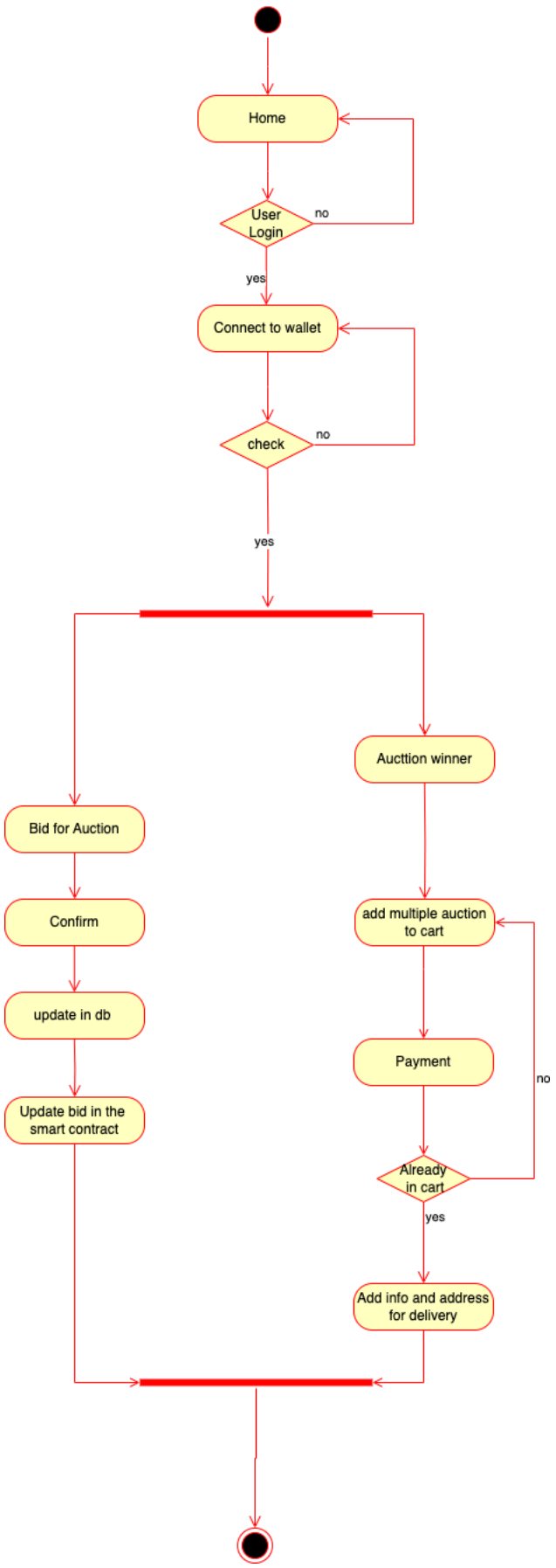
1.2.3. AD Home:


Home Page
Logged In
connect wallet
check
no
yes
Is Admin
yes
no
Set access to user
Set acces to admin

1.2.4. AD Admin:

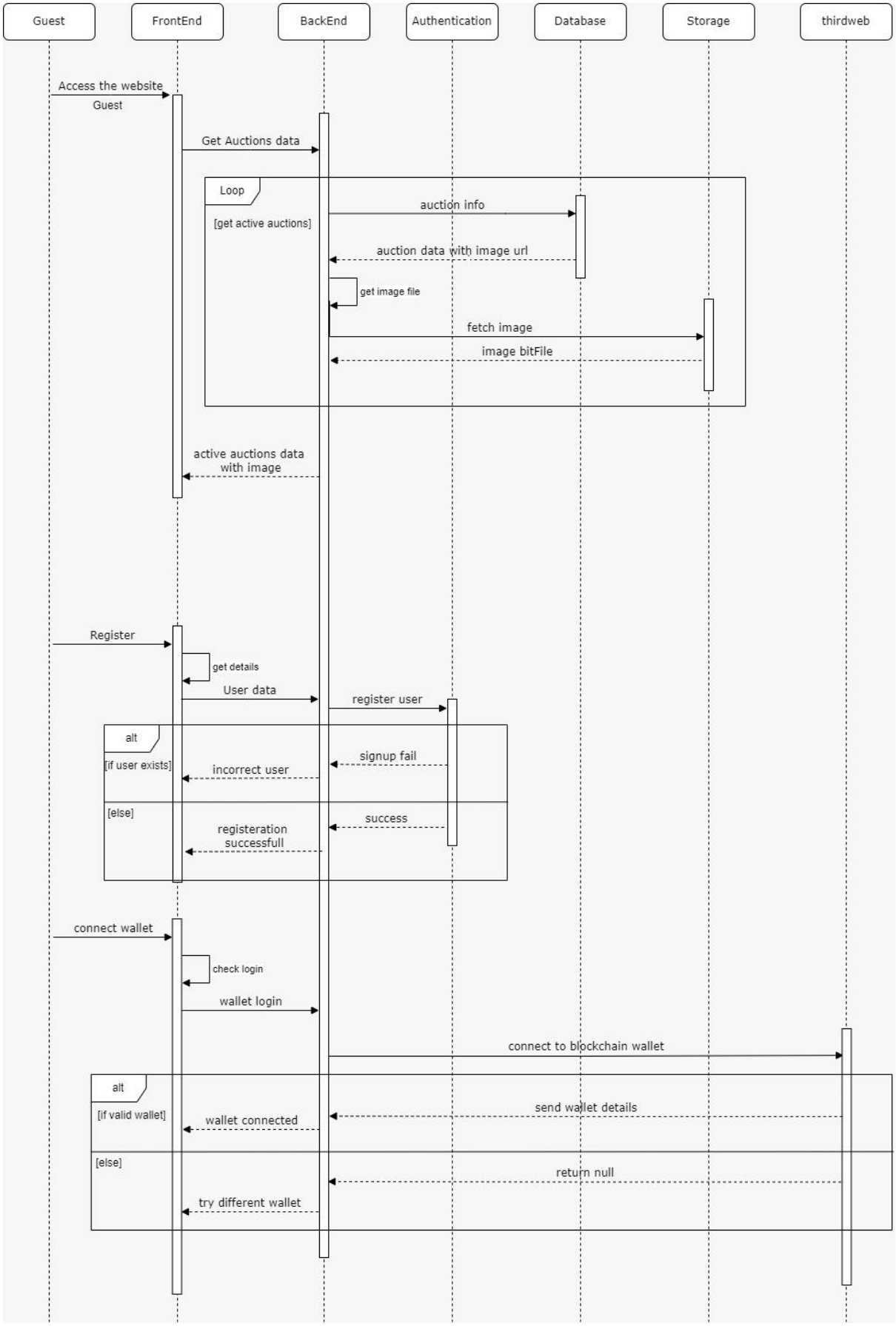

Home
is admin
no
yes
Connect wallet
check
no
yes
create Auction
Open Form
Add new auction details
upload image to storage
update to db
create new auction contract block
Delete Auction
Delete
delete data from db
end auction contract
Payments
Check Payment Status
Get User info of completed Payment

1.2.5. AD User:

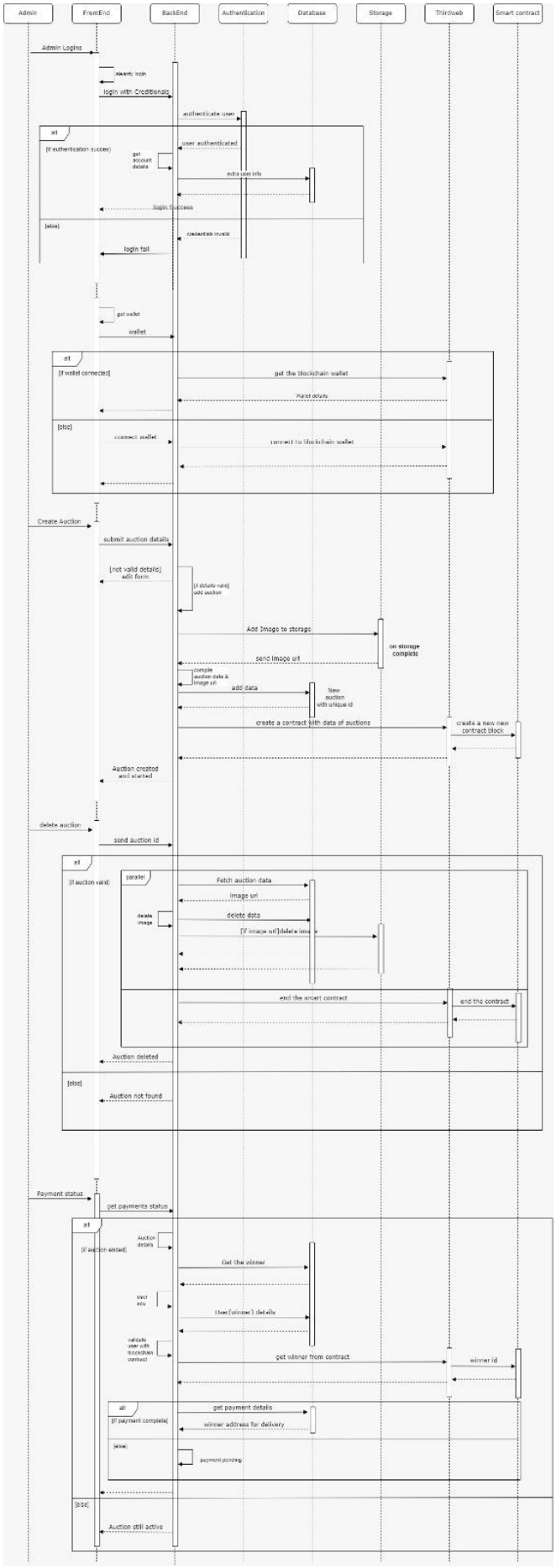

Home
User Login
no
yes
Connect to wallet
check
no
yes
Aucttion winner
Bid for Auction
Confirm
update in db
Update bid in the smart contract
add multiple auction to cart
Payment
no
Already in cart
yes
Add info and address for delivery

1.3. Sequence design

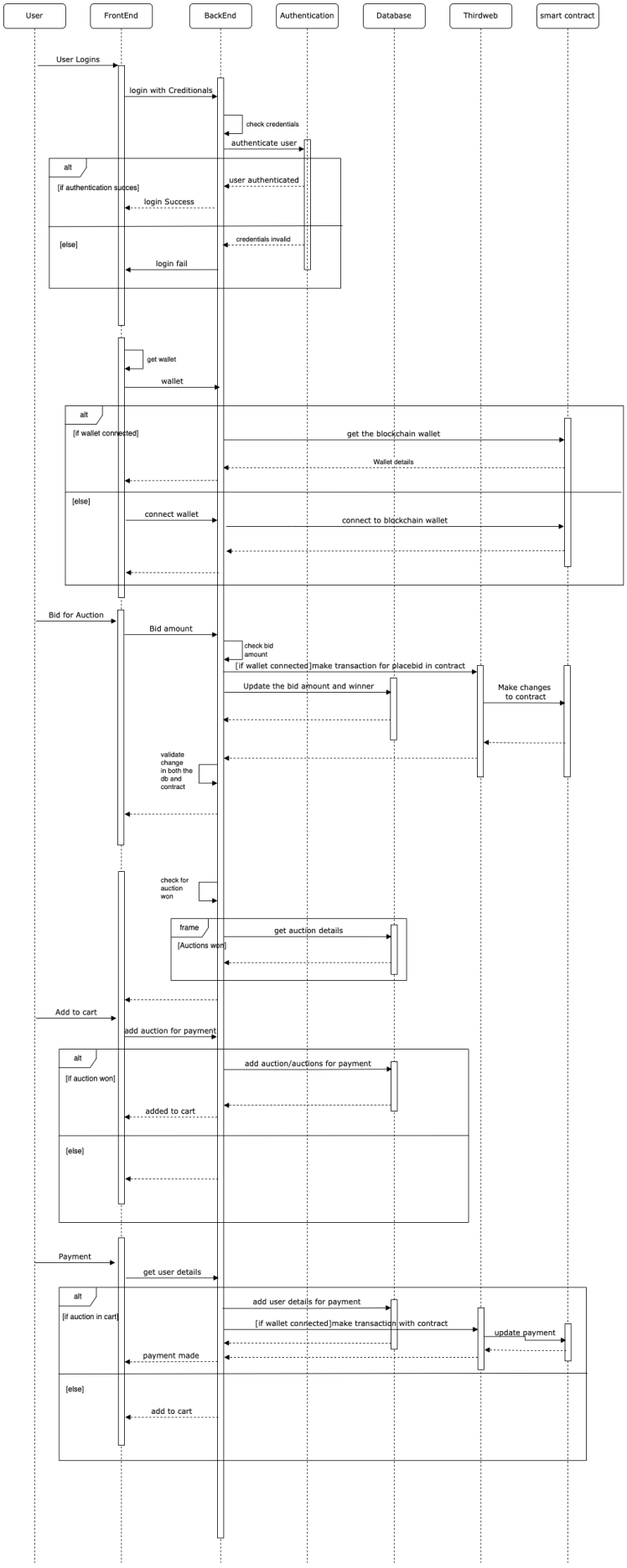
1.3.1. Auction Home:


Guest
FrontEnd
BackEnd
Authentication
Database
Storage
thirdweb
Access the website
Guest
Get Auctions data
Loop
[get active auctions]
auction info
auction data with image url
get image file
fetch image
image bitFile
active auctions data
with image
Register
get details
User data
register user
alt
[if user exists]
signup fail
incorrect user
[else]
success
registeration
successfull
connect wallet
check login
wallet login
connect to blockchain wallet
alt
[if valid wallet]
send wallet details
wallet connected
[else]
return null
try different wallet

1.3.2. Admin:


Admin
FrontEnd
BackEnd
Authentication
Database
Storage
Thirdweb
Smart contract
Admin Logins
Already login
login with Creditionals
authenticate user
alt
[if authentication succes]
user authenticated
get account details
extra user info
login Success
[else]
credentials invalid
login fail
get wallet
wallet
alt
[if wallet connected]
get the blockchain wallet
Wallet details
[else]
connect wallet
connect to blockchain wallet
Create Auction
submit auction details
[not valid details]
edit form
[if details valid]
add auction
Add Image to storage
on storage complete
send Image url
compile auction data & image url
add data
New auction with unique id
create a contract with data of auctions
create a new new contract block
Auction created and alerted
delete auction
send auction id
alt
[if auction valid]
parallel
Fetch auction data
image url
delete image
delete data
[if image url]delete image
end the smart contract
end the contract
Auction deleted
[else]
Auction not found
Payment status
get payments status
alt
[if auction ended]
Auction details
Get the winner
user info
User(winner) details
validate user with blockchain contract
get winner from contract
winner id
alt
[if payment complete]
get payment details
winner address for delivery
[else]
payment pending
[else]
Auction still active

1.3.3. User:

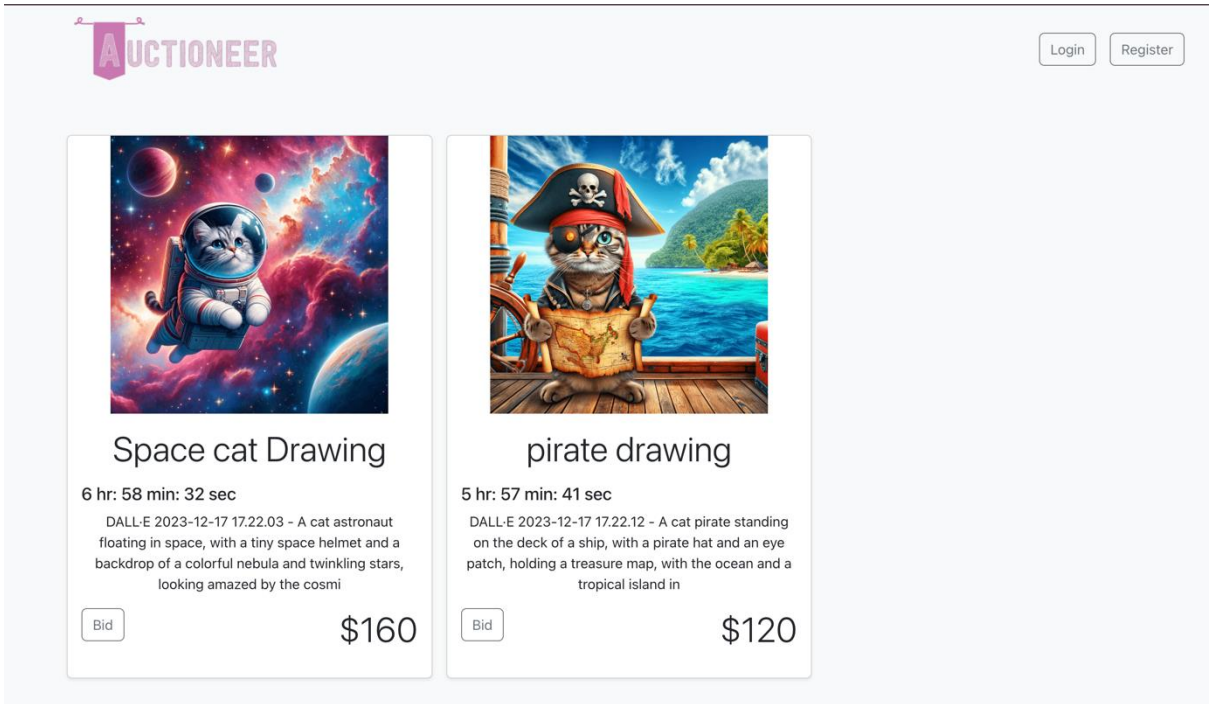

User
FrontEnd
BackEnd
Authentication
Database
Thirdweb
smart contract
User Logins
login with Creditionals
check credentials
authenticate user
alt
[if authentication succes]
user authenticated
login Success
[else]
credentials invalid
login fail
get wallet
wallet
alt
[if wallet connected]
get the blockchain wallet
Wallet details
[else]
connect wallet
connect to blockchain wallet
Bid for Auction
Bid amount
check bid amount
[if wallet connected]make transaction for placebid in contract
Update the bid amount and winner
Make changes to contract
validate change in both the db and contract
check for auction won
frame
[Auctions won]
get auction details
Add to cart
add auction for payment
alt
[if auction won]
add auction/auctions for payment
added to cart
[else]
Payment
get user details
alt
[If auction in cart]
add user details for payment
[if wallet connected]make transaction with contract
update payment
payment made
[else]
add to cart

Auctions Website

We have explained the implemented pages with GUI, their functionalities and explained some low-code functions implemented.

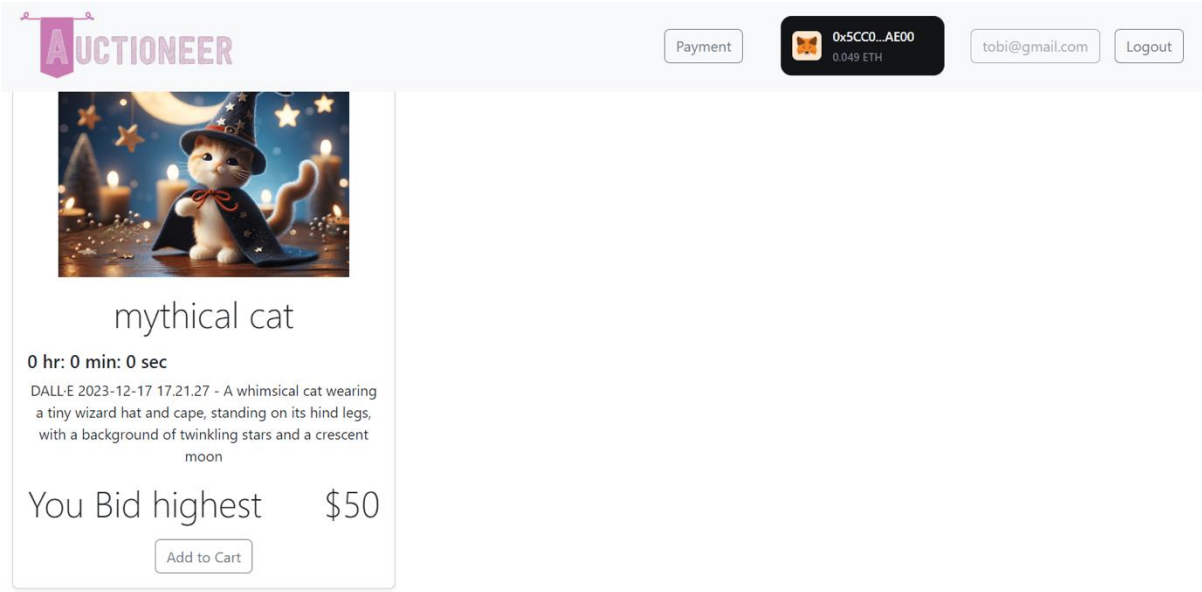
1. User Handle

1.1. Guest Home page



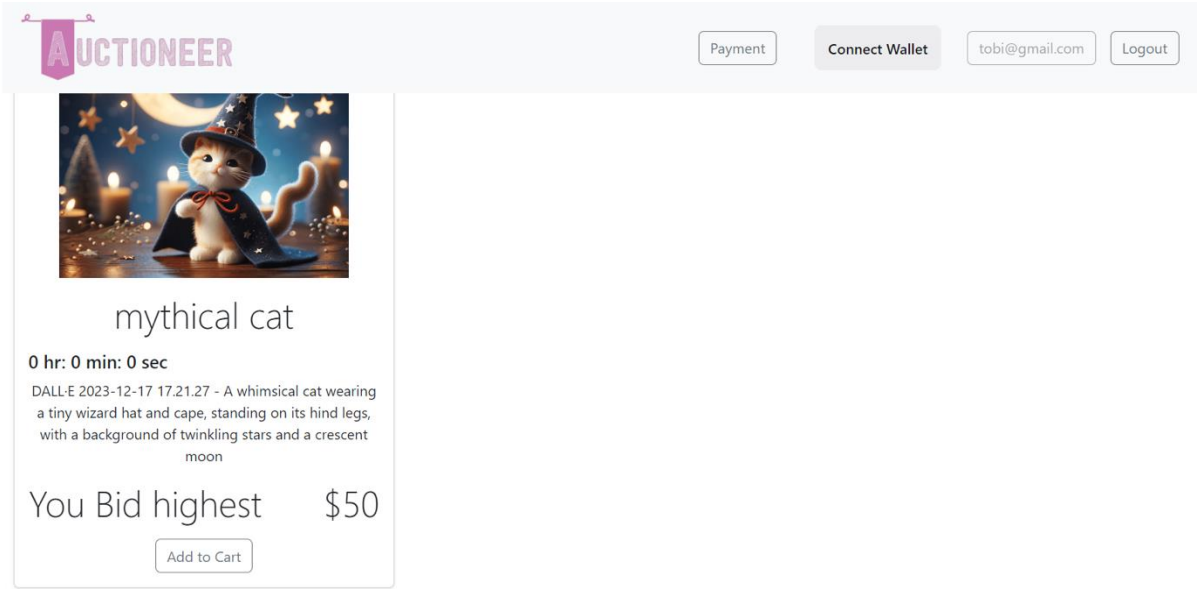
This is the page the guest will view for the first time visiting our website, User can try to signup, login to the website to access all the auctions and bid for them.

1.2. User Home page



Finally, this is the User view of home page for the Auctions that is visible to the user upon logging in to the website, user can place bid of their desired auctions, make a payment upon checkout.

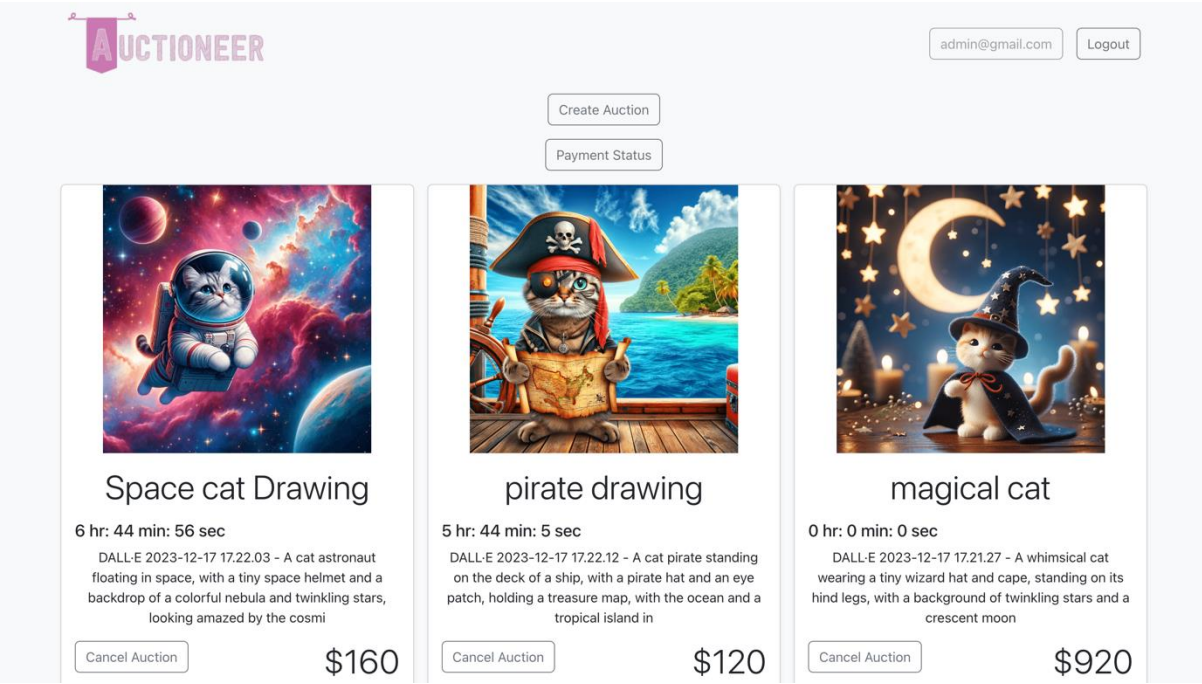
1.3. User Payment handle



This is the view of the home page when the user wins the bid after the time has been completed and the bid is ready for the user to add to cart and make a payment.

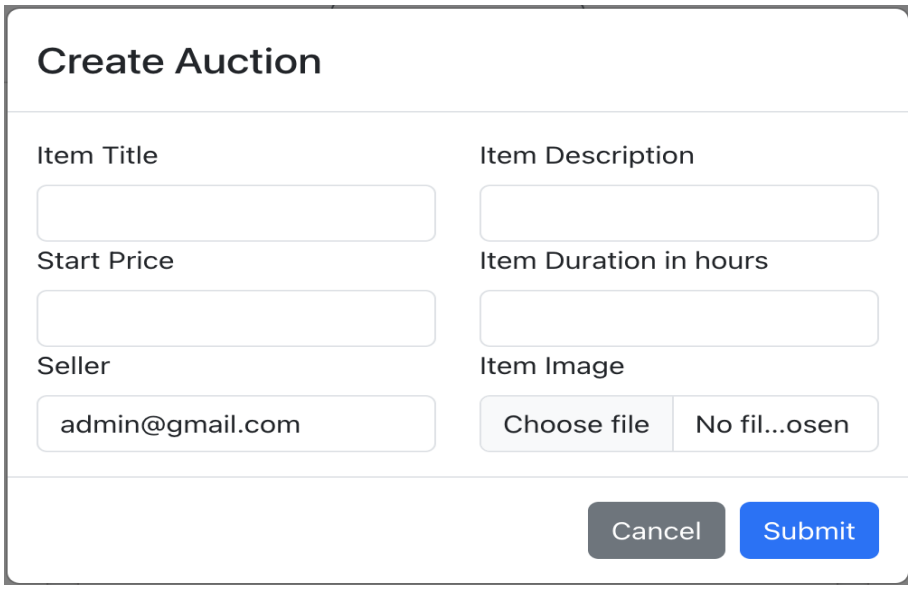
2. Admin Handle

2.1. Admin Home page



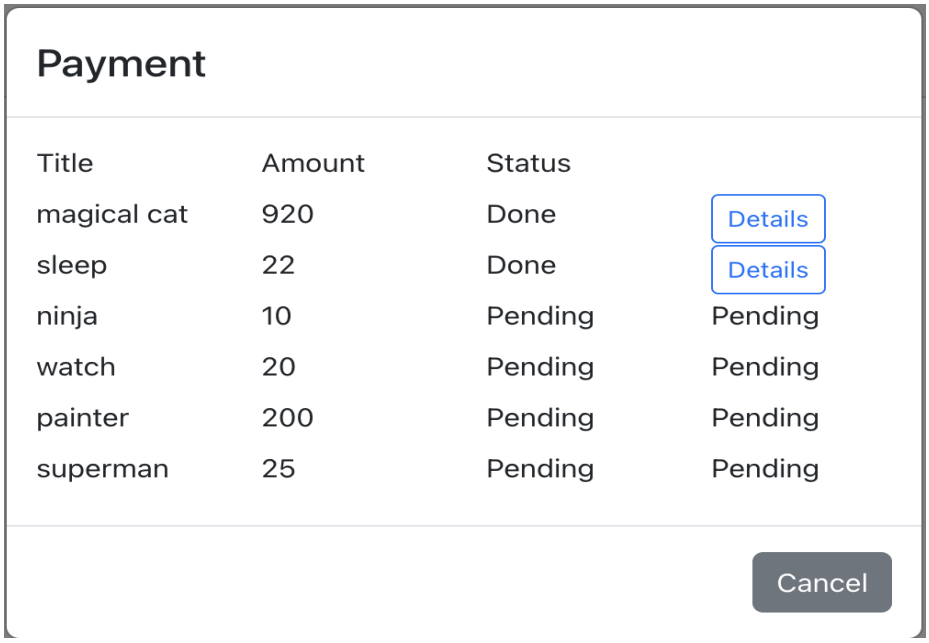
The Admin home page view where the admin can view his published auctions, delete them, and also create more auctions and check the corresponding status of the payments of the auctions that are already bid and purchased.

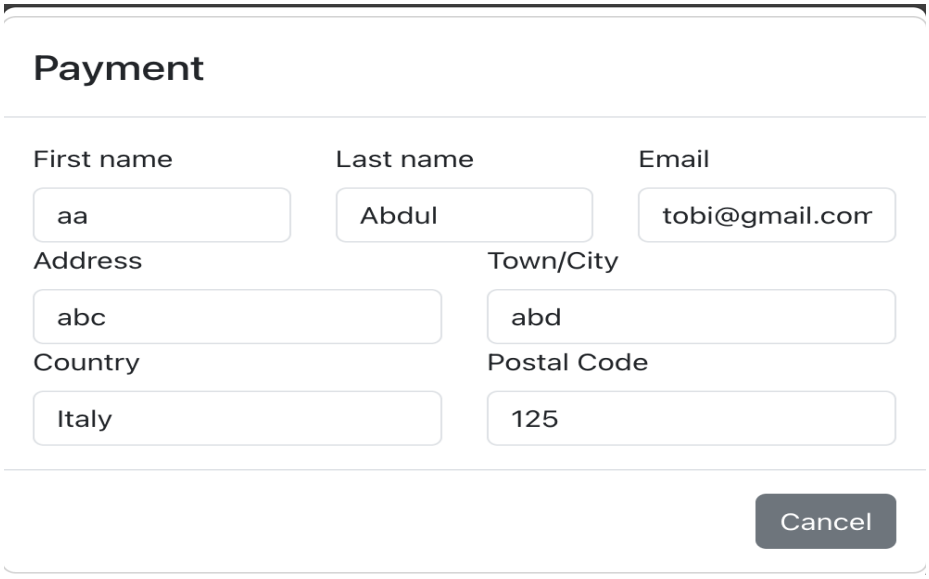
2.2. Admin Create Auction



The admin can create the auction by adding the corresponding details visible above to publish their auctions so that they are visible to the users. Admin needs to add the title, description of the auction, minimum price to start the bid from and the number of hours the auction is available, with the email id and also uploading the image file.

2.3. Admin Payment handle





The Admin payment handle allows the admin to monitor the status of the payment of their auctions and also view the details of the user that purchased the corresponding auction.

3. Code for API handling

There are some complex contexts which are used for handling the APIs used in this project.

First for the authentication, to check whether which type of user is logged in or not with the user details. If logged in, get the auctions the user had bid and etc.

```
export const AuthContext = createContext();
export const AuthProvider = ({ children }) => {
  const [currentUser, setCurrentUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const [globalMsg, setGlobalMsg] = useState('');
  const [ getRole , setRole] = useState(null);

  const register = (email, password) => {
    return createUserWithEmailAndPassword(authApp,email, password);
  };

  const login = (email, password) => {
    return signInWithEmailAndPassword(authApp, email, password);
  };

  const logout = () => {
    return signOut(authApp);
  };

  const bidAuction = (auctionId, price) => {
    if (!currentUser) {
      return setGlobalMsg('Please login first');
    }

    const db = collection(firestoreApp,'auctions');

    return updateDoc(doc(db,auctionId),{
      curPrice: price,
      curWinner: currentUser.email,
      curWinnerUid: currentUser.uid,
    });
  };

  const endAuction = (auctionId) => {
    const db = collection(firestoreApp,'auctions');

    return deleteDoc(doc(db,auctionId));
  };

  const addToCart = async (auctionId) => {
    const db = collection(firestoreApp,'user', currentUser.uid ,'cart')
    const cartState = await getDoc(doc(db,auctionId))
    if(!cartState.exists()){
      return setDoc(doc(db,auctionId),{paymentCompleted : 'false'})
    }
  }

  const removeFromCart = (auctionId) => {
    const db = collection(firestoreApp,'user', currentUser.uid ,'cart')
    return deleteDoc(doc(db,auctionId))
  }
```

```
    useEffect(() => {
      const interval = setTimeout(() => setGlobalMsg(''), 5000);
      return () => clearTimeout(interval);
    }, [globalMsg]);

    return (
      <AuthContext.Provider
        value={{
          currentUser,
          register,
          login,
          logout,
          bidAuction,
          endAuction,
          globalMsg,
          getRole,
          addToCart,
          removeFromCart,
        }}
      >
        {!loading && children}
      </AuthContext.Provider>
    );
  };
```

Below is the code for contract handling with the APIs.

```
const StateContext = createContext();

export const StateContextProvider = ({ children }) => {
  const { contract } = useContract('0xD7AB7554653DDD322b3596194aC4D77962036089');
  const { mutateAsync: addAuction } = useContractWrite(contract, 'addAuction');

  const address = useAddress();

  const createAuctionContract = async (id, name, desc, bid) => {
    try{
        await addAuction({args:[id,name,desc,bid]})
    }catch(error){
        console.log(error)
    }
  }

  const endAuctionContract = async (id) => {
    try {
        await contract.call('endAuction',[id])
        console.log("success")
    } catch (error) {
        console.log(error)
    }
  }

  const placebidContract = async (id, bid) => {
    try {
        await contract.call('placeBid',[id,bid])
        console.log("success")
    } catch (error) {
        console.log(error)
    }
  }

  return (
    <StateContext.Provider
      value={{
        address,
        contract,
        createAuctionContract,
        endAuctionContract,
        placebidContract
      }}
    >
      {children}
    </StateContext.Provider>
  )
}

export const useStateContext = () => useContext(StateContext)
```

We also have context for uploading the image to the firebase cloud storage

```
const UploadStorage = (data, setProgress, auctionContract) => {
    const data_r = data;
    const storageRef = ref(storageApp, data.itemImage.name);
    const collectionRef = collection(firestoreApp, 'auctions');
    const uploadTask = uploadBytesResumable(storageRef, data.itemImage);
    uploadTask.on(
        'state_changed',
        (snapshot) => {
            let percentage = (snapshot.bytesTransferred / snapshot.totalBytes) * 100;
            setProgress(percentage);
        },
        (err) => {
            console.log(err);
        },
        async () => {
            try {
                const imgUrl = await getDownloadURL(uploadTask.snapshot.ref)
                delete data_r.itemImage;
                const dockref = await addDoc(collectionRef, { ...data_r, createdAt:serverTimestamp(), imgUrl });
                await auctionContract(dockref.id,data.title,data.desc,data.curPrice)
                window.location.reload();
            }catch (error) {
                console.error('Error uploading data: ', error);
                alert('Error uploading data');
        }
        }
    );
};

export default UploadStorage;
```

There is also a configuration file which is needed to configure the firebase system which we will use for communication. There is also the solidity contract which we had used for making and deploying the smart contract.

```
// Use your own configs!
const firebaseConfig = {
  apiKey: "AIzaSyBrzXAIIyLP5DeJyALNrqF4FD6WNZZJemQ",
  authDomain: "auction-website-ab5d6.firebaseapp.com",
  projectId: "auction-website-ab5d6",
  storageBucket: "auction-website-ab5d6.appspot.com",
  messagingSenderId: "506409736189",
  appId: "1:506409736189:web:357dd8e8fe90f8d603ab94",
  measurementId: "G-TG403TZ82R"
};
const app = initializeApp(firebaseConfig)

export const firestoreApp = getFirestore(app);
export const storageApp = getStorage(app);
export const authApp = getAuth(app);
```

```
contract Auction {
    address public admin;

    struct AuctionItem {
        string id;
        string name;
        string description;
        uint startBid;
        address highestBidder;
        uint highestBid;
        bool isActive;
    }

    mapping(string => AuctionItem) public auctions;

    event AuctionCreated(string id, string name, uint startBid);
    event NewBid(string auctionId, address bidder, uint bidAmount);
    event AuctionEnded(string auctionId, address winner, uint winningBid);

    modifier onlyAdmin() {
        require(msg.sender == admin, "Only admin can perform this action");
        _;
    }

    modifier auctionExists(string memory _auctionId) {
        require(bytes(auctions[_auctionId].id).length != 0, "Auction does not exist");
        _;
    }

    modifier isActiveAuction(string memory _auctionId) {
        require(auctions[_auctionId].isActive, "Auction is not active");
        _;
    }

    constructor() {
        admin = msg.sender;
    }

    function addAuction(string memory _auctionId,string memory _name, string memory _description, uint _startBid) public onlyAdmin {
        require(bytes(auctions[_auctionId].id).length == 0, "Auction ID already exists");

        auctions[_auctionId] = AuctionItem(
            _auctionId,
            _name,
            _description,
            _startBid,
            address(0),
            0,
            true
        );

        emit AuctionCreated(_auctionId, _name, _startBid);
    }
```

Deployment

1.1. Project Setup

- Initiated a React project using create React App, structuring the project with reusable components to ensure maintainability and scalability.
- Rather than taking the raw code, we had compiled a ‘npm build’ for the deployment which returns the entire code excluding the files.

1.2. Firebase Configuration

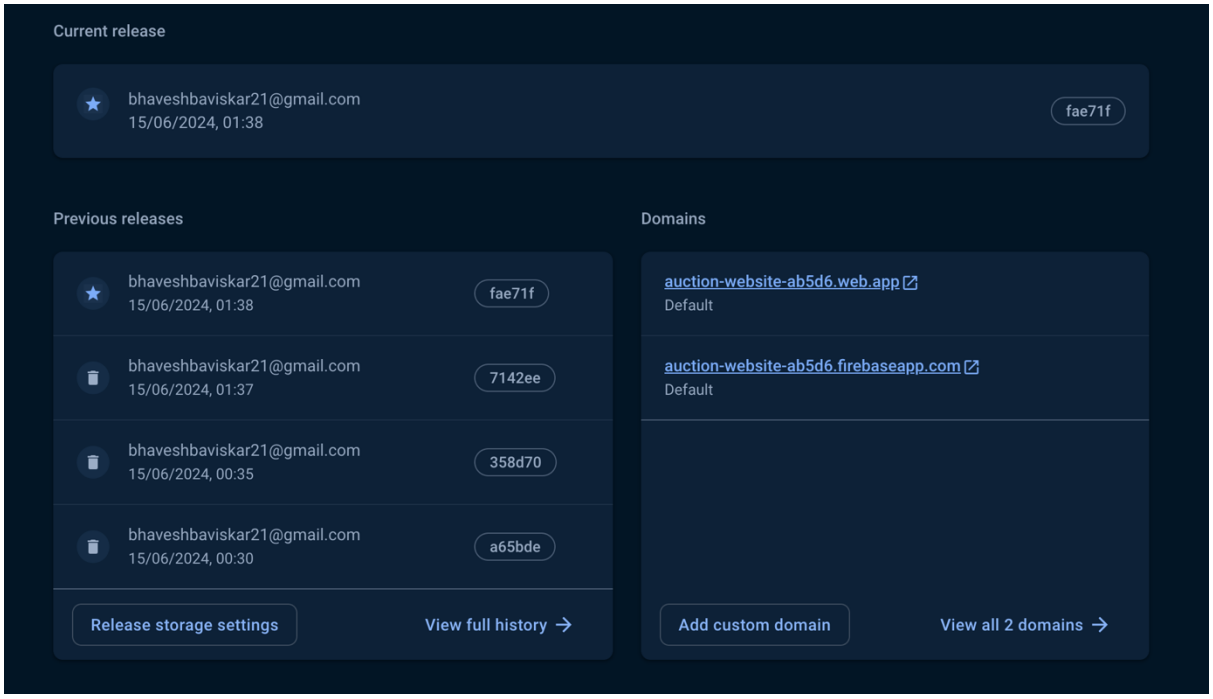
- Created a Firebase project in the Firebase console.
- Configured Firebase Hosting by installing the Firebase CLI and initializing the project with 'firebase init’ command.
- Setup Firebase Authentication to manage user sign-in and sign-up processes securely.

1.3. Development

- Developed key features such as user registration and login, item listing, auction creation, bidding system, and real-time updates using React's state management and Firebase's real-time database.
- Ensured responsive design using CSS media queries to cater to users on different devices.

1.4. Deployment

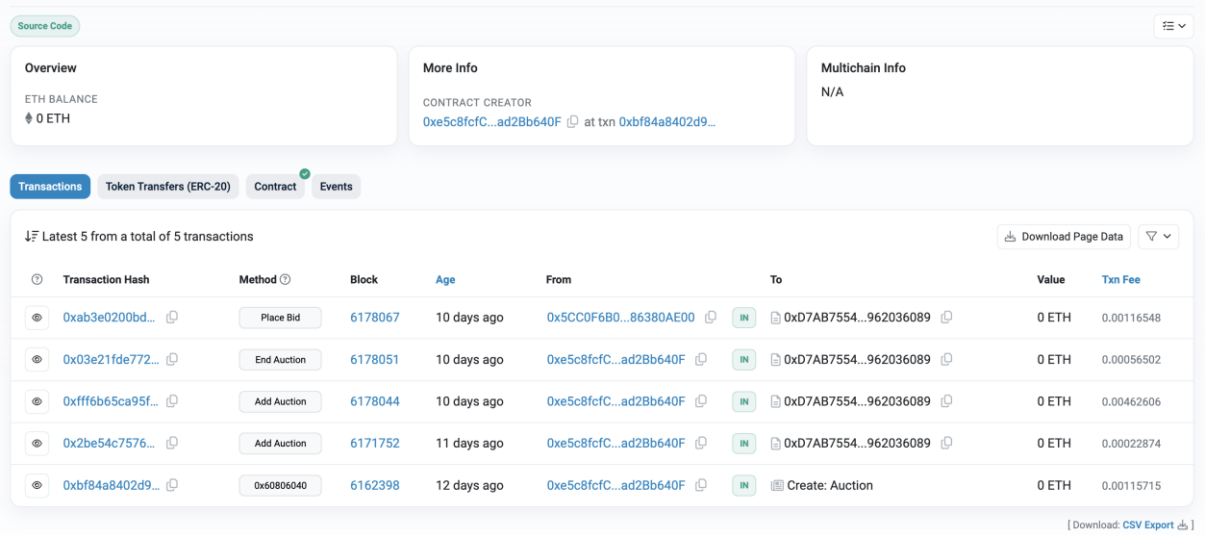
- Built the React application using the ‘npm run build’ command to generate a production-ready build.
- Deployed the build directory to Firebase Hosting using the Firebase CLI with the ‘firebase deploy’ command.
- Firebase Hosting offers fast, secure, production-grade web hosting for developers. It allows quick deployment of web apps to a global CDN with a single command. While optimized for static and single-page apps, it can also work with Cloud Functions or Cloud Run to host dynamic content and microservices.
  - We first initialize the project directory for deployment:
    - $ firebase init hosting
  - Then we modify the build file in the directory we initialized.
  - We then deploy the whole directory to the firebase:
    - $ firebase deploy
  - Finally we run the URL in the browser which we got from the firebase after deployment.



- The only necessary thing that should be deployed and hosted is the react build file which we deploy with the firebase deployment. As the firebase database and firebase storage are a cloud system which user interacts with an API integrated in the build file, we don’t need to deploy it in the system separately. Also the authentication is a API model so it can used from any platform.
- The further deployments can be made with just modifying the code and changing the deployed build file with the new build file. These functionalities work when the system is online by also keeping the downtime minimum.

1.5. Deployment of Blockchain

- we have implemented blockchain smart contracts using Third Web for Ethereum wallets. This integration enables secure and transparent transactions for auction participants. By leveraging Third Web's capabilities, we ensure that bidding and transactions are handled directly through blockchain technology, enhancing reliability and trust in the auction process. This setup not only provides a seamless user experience but also aligns with modern standards, ensuring that all interactions are verifiable and immutable.
- First, we created the solidity contract which has all our requirements for the smart contract which we will deploy. Once the smart contract is finalized, we deploy it with the third web to the sepolia network for our website to run and use smart contract.
- We need to run the command 'npx thirdweb deploy' from our root directory of the solidity contract. This will open the interface on the website third web which we select the type of network we want to deploy for the contract and the cost of the deployment.
- Once we have done with the deployment, we just need to execute our code to work with the deployed contract.



1.6. Testing

1.6.1. Verification tests

FR-1: User Authentication Management:

- Register user
  - Passed: Verify that all required fields are present, such as email, password.
  - Passed: Verify that user can successfully register with valid information.
  - Passed: Checking if a user can register with valid data only in mandatory fields Passed: A user get registered should be allowed to login to the application.

- Passed: A user tries to register an existing email then an error message should get displayed.
- Passed: Verify if the registration form has all the fields required by the requirement document.
- Passed: Verify that a user can navigate to Login if they have an account.

- Login User
  - Passed: Verify that all required fields are present, such as email, password.
  - Passed: Verify that a user can log in with valid credentials.
  - Passed: Verify that a user cannot log in with invalid credentials.
  - Passed: Verify that a user cannot log in with invalid email and password.
  - Passed: Verify that a user can navigate to Register if they don't have an account.

- Connect to wallet
  - Passed: Verify user details to the blockchain wallet.
  - Passed: Verify if the details of the blockchain wallet are authorized.
  - Passed: Verify that the wallet has successfully been connected to the account.

FR-2: Bid Management:

- Choosing the auction
  - Passed: verify that inly active auction are shown
  - Passed: verify that only auction not bid by user can be bided

- Bid for auction
  - Passed: verify that bid value change is update and modified across every thing
  - Passed: verify that the auction won by user are not bided by same user

- Add to cart
  - Passed: Verify that users can add products to their cart.
  - Passed: Ensure the correct product is added to the cart.
  - Passed: Check if the cart updates with the newly added product.
  -

FR-3: Payment Management:

- Payment for the bid
  - Passed: Verify that users can proceed to checkout from their cart.
  - Passed: Ensure the checkout process captures all necessary user information.
  - Passed: Check if the order is successfully placed and confirmed after checkout.

- Payment through Blockchain wallet

  - Passed: Verify that users can make payments using a wallet.
  - Passed: Ensure the payment process securely integrates with the blockchain.
  - Passed: Check if users receive a payment confirmation after successful transaction.

FR-4: Admin Authentication Management:

- Admin Login
  - Passed: Verify that all required fields are present, such as email, password.
  - Passed: Verify that the admin can log in with valid credentials.
  - Passed: Verify that the admin cannot log in with invalid credentials.

- Passed: Verify that the admin cannot log in with invalid email and password.
- Passed: Verify that the admin can navigate to Register if they don't have an account.

- Admin Get Blockchain wallet
  - Passed: Verify if admin if connected to wallet for auction creation.
  - Passed: Verify if the wallet is connected for ending the auction.

FR-5: Admin Auction Management:

- Create Auction
  - Passed: Verify is the wallet is connected.
  - Passed: Verify if the auction is not already exists.
  - Passed: Verify if is admin for the creation.

- Contract creation
  - Passed: Verify is the contract is created on creating new auction.
  - Passed: Verify is auction is available to all.

- Admin wallet contract handling
  - Passed: Verify is the contract is deletes on creating new auction.
  - Passed: Verify is auction is not modified if not active.

- Delete Auction
  - Passed: Verify auction details are deleted from database is deleted.
  - Passed: Verify if the admin is taking the action in deletion.

- Uploading Auction image
  - Passed: Verify if the image is of correct type.
  - Passed: Verify if the image if of correct bytes.
  - Passed: Verify is image is correctly uploaded.
  - Passed: Verify get the image URL if completed.

FR-6: Admin Payment Management:

- Payment status
  - Passed get payment status if payed by user.
  - Passes get the details of the auction won and their winner details.
  - Passed verify the auction winner with the blockchain smart contract.
  - Passed verify that the auction has ended correctly.

A. User Testing: Engaged a diverse group of users to interact with the website, covering various demographics and levels of technical proficiency. Users were provided with specific tasks to

perform, such as registering an account, creating an auction, placing bids, and navigating through the site.

B. Cross-Platform testing: Tested the website on multiple platforms and devices, including desktop (Windows, macOS, Linux) and mobile devices (iOS, Android). Various web browsers were used for testing, to ensure compatibility and consistent performance.

C. Live Bidding tests: Conducted simulated live auctions with multiple users participating simultaneously to test the real-time bidding functionality. Focused on ensuring that bids were updated in real-time without delays and that all users received accurate and timely notifications of the bid status.

1.7. User Requirements addressed

- User-friendliness was a top priority, and we designed the application with an intuitive interface to cater to users of varying technical expertise. Ensuring accessibility across different devices was essential, allowing users to work seamlessly from desktops and tablets, thereby enhancing productivity. Performance was a critical requirement, and we optimized the application to be fast and responsive, meeting user expectations for a high-performing tool. Security was paramount due to the sensitive nature of the organizational data. We implemented stringent security measures to safeguard data and ensure that only authorized personnel could access specific features and information. Reliability was also key; by focusing on thorough testing and monitoring, we ensured the application was stable and available when needed. Finally, we addressed the need for ongoing support and training by setting up dedicated channels and continuous training programs to keep users updated and proficient with new features and improvements.

Conclusion

The auctions website, developed using React and deployed on Firebase Hosting, has undergone rigorous testing to ensure robustness, usability, and performance. Comprehensive user testing, cross-platform compatibility checks, and live bidding simulations were conducted to identify and resolve issues. The testing process led to significant improvements in user authentication, auction management, and real-time bidding functionality. Performance enhancements, including optimized loading times and scalability, were implemented to handle high traffic conditions. User feedback drove refinements in the interface and usability, ensuring an intuitive and seamless experience across desktop and mobile devices. The website is now well-equipped to deliver a reliable, engaging, and efficient auction platform for users.