

Bid 4 Good Documentation

The Auction Website Project aimed to create an online platform for auctions and marketplace sales. The project was initiated with the objective to develop a user-friendly and secure website that would enable verified sellers to create two types of auctions - live and blind - as well as sell items on the marketplace. The website was also designed to allow buyers to participate in auctions and make purchases.

Dependencies

Client Dependencies

Dependency	Description
@formkit/vue	A Vue.js plugin for building powerful and flexible forms. Used for developing flexible forms in the entire application.
@fortawesome/free-regular-svg-icons	A set of free SVG icons for use with the @fortawesome/vue-fontawesome plugin.
@kyvg/vue3-notification	A Vue.js plugin for displaying notifications. These are used application wide for showing notifications as growl messages.
Axios	A JavaScript library for making HTTP requests. This is responsible for carrying out all the communication between BackEnd and FrontEnd via HTTP Requests.
Bootstrap	A popular CSS framework for building responsive websites. Used for making flexible, and easy designing.
socket.io-client	A JavaScript library for using Socket.IO on the client side. This is being utilized in Live Bidding feature, where we are carrying out the communication with the help of sockets. They help in broadcasting our information.
Vue	A progressive JavaScript framework for building user interfaces. We are using VUE 3 for developing our Front End web application.
vue3-carousel	A Vue.js plugin for creating carousels. These are used widely in Home Page, Bidding pages, etc. They are responsible for adding a carousel-like effect at the necessary places in application.
vuex	A state management pattern + library for Vue.js applications.
vuex-persistedstate	A plugin for vuex that automatically persists and rehydrates the state on page reload. The above are responsible for carrying out state management in the Vue Application and storing the information in session storage so that it won't get lost if the user hits the refresh button.

Setting up Client-Side Server

Above Dependencies are Required to run Client-Side Server.

Prerequisites

Before you begin, ensure that you have the following:

- Node.js installed on your machine
- Basic knowledge of JavaScript, HTML, and CSS

Recommended IDE Setup

VSCode + Volar (and disable Vetur) + TypeScript Vue Plugin (Volar).

Instructions to run client-side locally:

- Move to the FrontEnd directory by running `cd Front End`.
- Install the Vue CLI (Command Line Interface) globally on your machine using NPM.
- `npm install -g @vue/cli`
- Install all the core dependencies
`npm install`
- To run the development server, use the following command: `npm run dev`
- This will the local development server at <http://localhost:5173/>

Server Dependencies

Dependency Name	Why it is Used
@types/chai	Provides TypeScript type definitions for the chai assertion library
@types/express	Provides TypeScript type definitions for the express framework
@types/node	Provides TypeScript type definitions for the Node.js runtime environment
nodemon	A tool that monitors changes in the code and automatically restarts the server
typescript	A language that extends JavaScript by adding type annotations
@types/mocha	Provides TypeScript type definitions for the mocha testing framework
@types/sinon	Provides TypeScript type definitions for the sinon mocking library
express	A popular Node.js web application framework

Setting Up Node Server

To operate the Bidforgood application server, you must install the **necessary** dependencies and **Node.js** version 8.15.0 or later, which you can get from <https://nodejs.org/en>. The steps to launch the server locally are as follows:

1. Clone the project from git.cs.dal.ca/courses/2023-winter/csci-5308/group05.git.
2. Go to the Server directory by running the command: `cd /server/`.
3. Install the required dependencies by running the command: `npm install`.
4. Compile the TypeScript code into JavaScript by running the command: `tsc -w`.
5. Start the server by running the command: `npm start`.
6. To execute a cron job script (which is necessary to send emails) : `npm run cron`.
7. The server will be accessible at <http://localhost:3000/>.
8. Below are the instructions for deploying the application on a virtual machine.

Build Deployment Instructions

1. Virtual Machine Configuration (Linux):

These are the necessary steps that need to be done in order to make the code

- a. Node version (v16.19.1 or above)
- b. Npm version (v7.0 or above)
- c. Run Following commands on VM:
 - i. `npm install -g typescript@5.0.2`
 - ii. `npm install -g nodemon@2.0.22`
 - iii. `npm install -g ts-node@10.9.1`
- d. Setup script provided under the `vm_scripts` folder on VM:
- e. Generate SSH key for VM
- f. Export SSH private key to CICD variables under "VM_PRIVATE_KEY"

2. Add repository to GitLab:

(Note: repository must have ".gitlab-ci.yml")

After adding the project to GitLab, git lab will run the pipeline and make connection with the VM and deploy it on them VM by running the scripts configured during configuration process.

Phases of automation for deployment:

The CI/CD pipeline is a sequence of steps that automates the building, testing, and deployment of software. It consists of five stages: Build, Test, Quality, Deploy, and Post-Deploy.

1. Continuous Integration (CI):

- build
- test
- quality

Build: This is the build stage, and it will have multiple jobs frontend-build, backend-build where code will be checked for compile and connection.

Test: This phase will start the unit testing and integration testing of the uploaded code, and it will make sure that every code is functioning Properly.

Quality: This job will run the and check if the quality of the code is proper or not.

2. Continuous Deployment (CD):

- deploy
- Post-deploy

Deploy: This job will connect to the Virtual Machine and upload the code from repository by making a .zip file for relevant job

Post-deploy: It will run the job for starting a script on the VM which will run the cron-job for our project “live auction”. That script will recursively check every minute, if auctions are over then sending an email to winner along with the payment.

3. Demo Deployed Application:

After the successful job running the website will be automatically deplo on the web-application on the Virtual Machine.

- VM HOST: csci5308vm5.research.cs.dal.ca
- Frontend: <http://csci5308vm5.research.cs.dal.ca/5173>
- Backend: <http://csci5308vm5.research.cs.dal.ca/3000>

Usage scenarios

The application is designed to cater to three types of users:

1. Buyers
2. Sellers
3. Administrator

Use Case: User Registration

The screenshot shows a web browser window with the address bar displaying 'csci5308vm5.research.cs.dal.ca:5173/reg-seller'. The page features a blue header with the 'Bid4Good' logo. Below the header is a 'Registration' form with the following fields and sections:

- First Name**: Text input field.
- Last Name**: Text input field.
- Phone Number**: Text input field.
- Date of birth**: Text input field with a date picker icon, placeholder 'dd/mm/yyyy'.
- Photo of Government ID**: File upload area with a placeholder 'No file chosen' and instructions: 'Upload a government approved ID such as Driving License or Passport. Only .pdf, .jpg, .jpeg, .png files allowed'.
- Address**: Text input field.
- State**: Dropdown menu with 'Select a State'.
- City**: Dropdown menu with 'Select City'.
- Postal Zip Code**: Text input field with format instructions: 'format: a1b-2c3 | a1b2c3 | a1b 2c3'.
- User Credentials**:
 - Email**: Text input field.
 - Confirm Password**: Text input field with placeholder 'Re-Enter Password'.

Primary Actors: Buyer, Seller

Stakeholders: • User: The person who wants to register on the platform. • Platform: The system that provides user registration functionality.

Preconditions: • The user has access to the platform. • The user has not registered with the same email address on the platform.

Main Success Scenario:

1. The user accesses the registration page of the platform.
2. The user enters their personal information, such as name, email address, password, Home Address, Date of birth, firstName, lastName, dateOfBirth, , Buyer or Seller, phone, address, cityName, provinceName, govtId, and postalCode.

3. The user submits the registration form.
4. The platform verifies that the email address must still be registered.
5. The platform creates a new user account with the provided information.
6. The platform sends a welcome email to the user's email address.
7. The platform's admin dashboard verifies the user's information and activates their account.
8. The platform displays a success message to the user and redirects them to the login page.

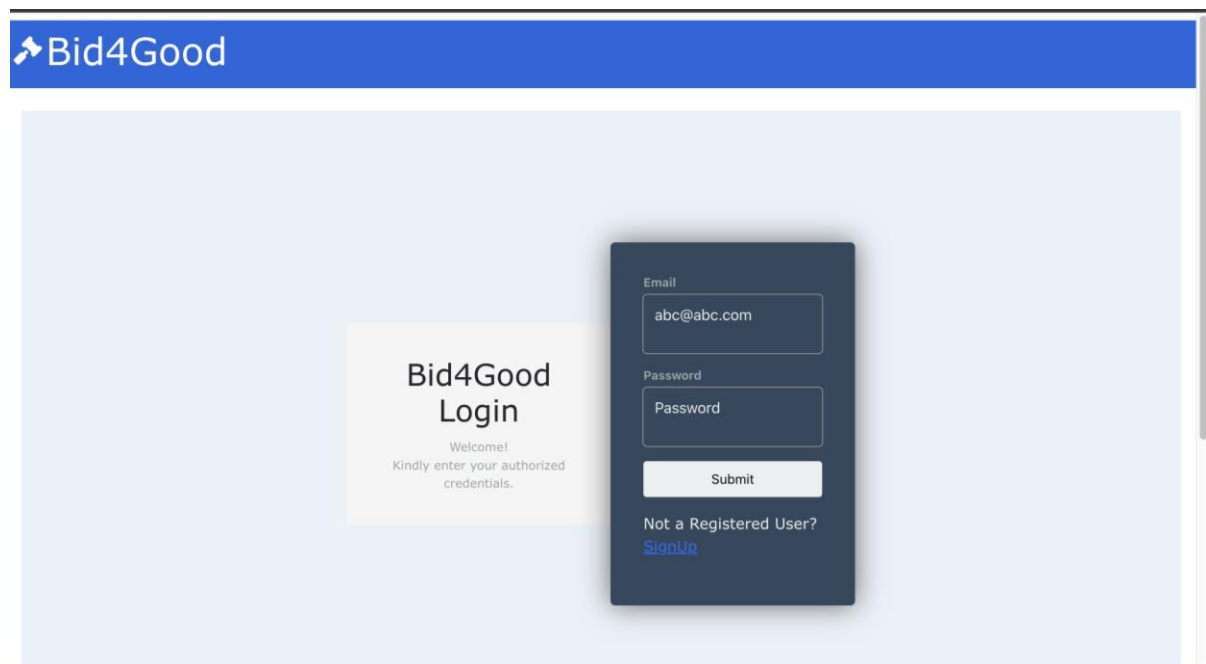
Post-conditions:

- The user's account is created and activated by Admin on the platform.
- The user can now log in to the platform using their email and password.

Exceptions:

- If the email address is already registered, the platform displays an error message to the user and prompts them to enter a different email address.
- If the user enters invalid information, the platform displays an error message and prompts them to correct the input.

Use Case: User Login



Primary Actors: Buyer, Seller

Stakeholders:

- User: The person who wants to log in to the platform.
- Platform: The system that provides user login functionality.

Preconditions:

- The user has an active account on the platform.
- The user has entered the correct login credentials.

Main Success Scenario:

- The user accesses the login page of the platform.
- The user enters their email address and password.
- The user clicks on the "login" button.
- The platform verifies the user's credentials.
- The platform grants access to the user's account.
- The platform redirects the user to the dashboard page.

Post-conditions:

The user is logged in to the platform and can access their account features.

Exceptions:

If the user enters an incorrect email or password, the platform displays an error message and prompts the user to enter the correct credentials.

Use Case: Adding an Auction by a Seller

Add Bid Details

Name Of Offering: Car

Estimated Value: 1200

Address: Dalhousie University

Start Date Of Auction: 10/04/2023

Start Time Of Auction: 05:58 PM

End Date Of Auction: 10/04/2023

End Time Of Auction: 05:58 PM

Bid Type: Live Bidding

State: British Columbia

City: Anmore

Postal Zip Code: a2b2c4

Photo of Items: car4.png

Description: Vintage car for sale

Primary Actor: Seller

Stakeholders:

- Seller: The user who wants to add an auction on the platform.
- Buyer: The user who wants to bid on the auction.
- Platform: The system that provides auction functionality.

Preconditions:

- The seller is registered, verified, and logged in to the platform.
- The seller has the information about the product they want to auction.

Main Success Scenario:

1. The seller accesses the "Add Auction" page of the platform by clicking the hammer button on the navbar.
2. The seller fills in the required fields such as the Name, Description of the Item, starting price, starting Time, ending Time, auction type (blind, live auction, or normal sell), address, city name, province name, postal code, and Image.
3. The seller submits the auction form.
4. The platform validates the auction form data and fills in all required fields.
5. The platform saves the auction details and displays a success message to the seller.
6. The auction becomes available for the buyers to bid.

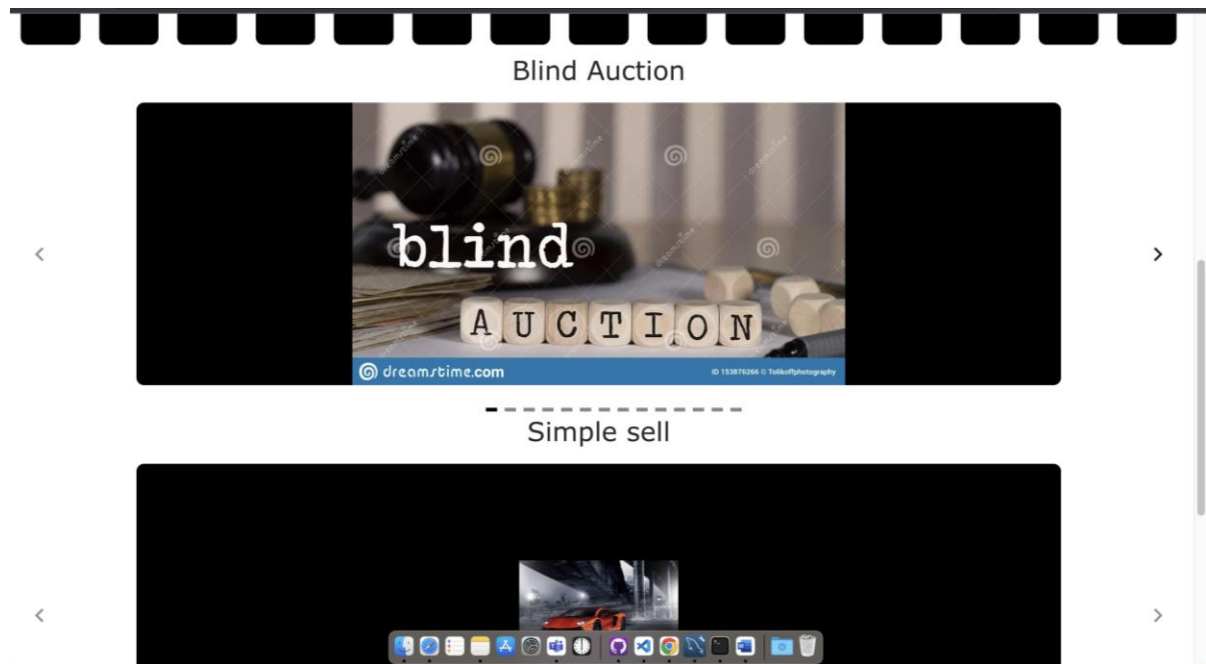
Post-conditions:

- The auction is added to the platform and available for the buyers to bid.
- The seller can view the details of the auction on the platform.

Exceptions:

- If the seller enters invalid information, the platform displays an error message and prompts them to correct the input.
- If the seller tries to add an auction with incomplete or missing information, the platform displays an error message and prompts them to fill in all required fields.

Use Case: Browsing Available Auctions on Homepage



Primary Actor: User

Stakeholders:

- User: The person who wants to browse available auctions on the platform.
- Seller: The user who has listed the auction on the platform.
- Platform: The system that provides auction functionality.

Preconditions:

- The user is logged in to the platform or browsing as a guest.
- There are auctions listed on the platform.

Main Success Scenario:

1. The user visits the homepage of the platform.
2. The platform displays a list of available auctions, categorised as blind, live, or normal sell items.
3. The user can browse any of the categories to view auctions in that category.
4. The user can use the search bar to search for auctions available within a particular time frame by entering the start and end times.
5. The platform displays a list of available auctions that match the search criteria, showing details such as the name, image, starting bid price, and time remaining for the auction.
6. The user can click on any auctions to view more details.

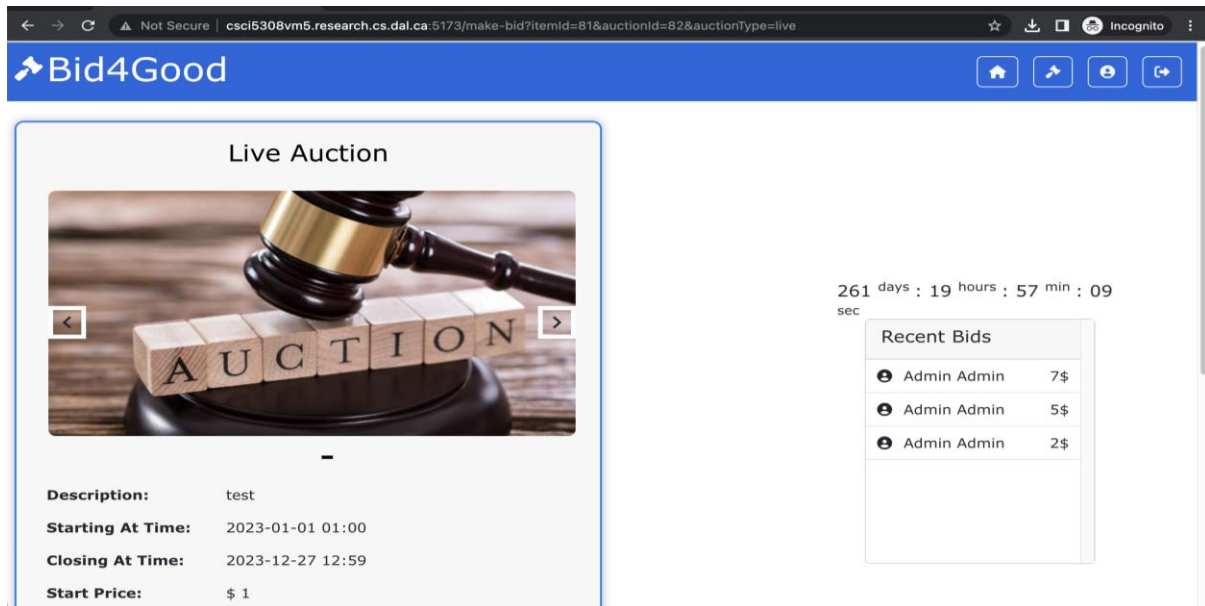
Post-conditions:

- The user can view available auctions on the platform and search for auctions within a particular time frame.

Exceptions:

- If no auctions are available on the platform, the platform displays a message indicating no auctions to display.
- If the user enters invalid search criteria, the platform displays an error message and prompts them to enter valid search criteria.
- If the platform experiences technical issues or server errors, the platform displays an error message and prompts the user to try again later.

Use Case: Participating in a Live Auction



Primary Actor: User

Stakeholders:

- User: The person who wants to participate in a live auction on the platform.
- Seller: The user who has listed the auction on the platform.
- Platform: The system that provides auction functionality.

Preconditions:

- The user is logged in to the platform.
- The platform has a live auction currently running.

Main Success Scenario:

1. The user navigates to the live auction page on the platform.
2. The platform displays information about the live auction, including the item being auctioned, the current highest bid, and the time remaining for the auction.
3. The user can click the "Join Auction" button to participate in the live auction.
4. The platform opens a bidding panel for the user to enter their bid.
5. The user enters their bid and clicks the "Submit Bid" button.
6. The platform validates the bid and updates the highest bid displayed on the screen if the user's bid is higher than the current highest bid.
7. The user is prevented from entering another bid for 10 seconds to prevent spamming the system.

8. The platform displays a list of current bids on the right-hand side of the screen, showing the username and the amount of the bid.
9. The auction continues until the time runs out.
10. The platform determines the highest bidder and displays their username as the winner of the auction.
11. The platform sends an email to the winner with a "Pay" button.
12. The winner clicks the "Pay" button, which redirects them to the payment page.
13. The winner completes the payment process.

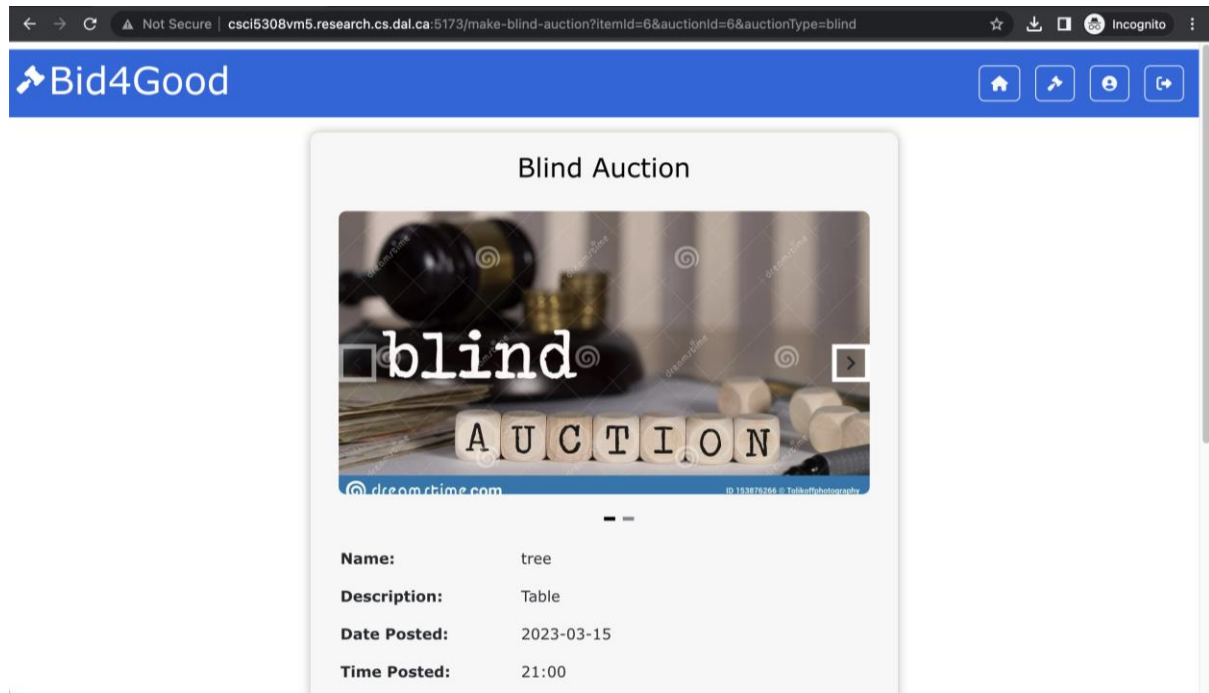
Post-conditions:

- The highest bidder has won the live auction and completed the payment process.

Exceptions:

- If the user enters an invalid bid, the platform displays an error message and prompts the user to enter a valid bid.
- If the user attempts to enter a bid before the 10-second restriction is lifted, the platform displays an error message and prompts the user to wait before entering another bid.
- If there are technical issues or server errors, the platform displays an error message and prompts the user to try again later.

Use Case: Participating in a Blind Auction



Primary Actor: User

Stakeholders:

- User: The person who wants to participate in the blind auction on the platform.
- Other bidders: Users who have also placed their bids in the blind auction.
- Seller: The user who has listed the product for auction on the platform.
- Platform: The system that provides the blind auction functionality.

Preconditions:

- The user is logged in to the platform.
- There is a blind auction available on the platform.
- The user has sufficient funds or payment methods to place the bid.

Main Success Scenario:

1. The user visits the blind auction page on the platform.
2. The platform displays the auction details, such as the name of the product, the starting bid price, and the time remaining for the auction.
3. The user enters the amount of their bid in the bidding field.
4. The platform checks if the bid amount is higher than the current highest bid.
5. If the bid amount is higher, the platform stores the bid amount and displays a message indicating that the user has placed the bid.

6. Other bidders can also place their bids until the auction ends.
7. After the auction ends, the platform generates the winner based on the highest bid placed (which is hidden from all bidders).
8. The platform sends an email to the winner with a payment button and redirects them to the payment page.
9. The user clicks on the payment button and is directed to the payment page to complete the transaction.

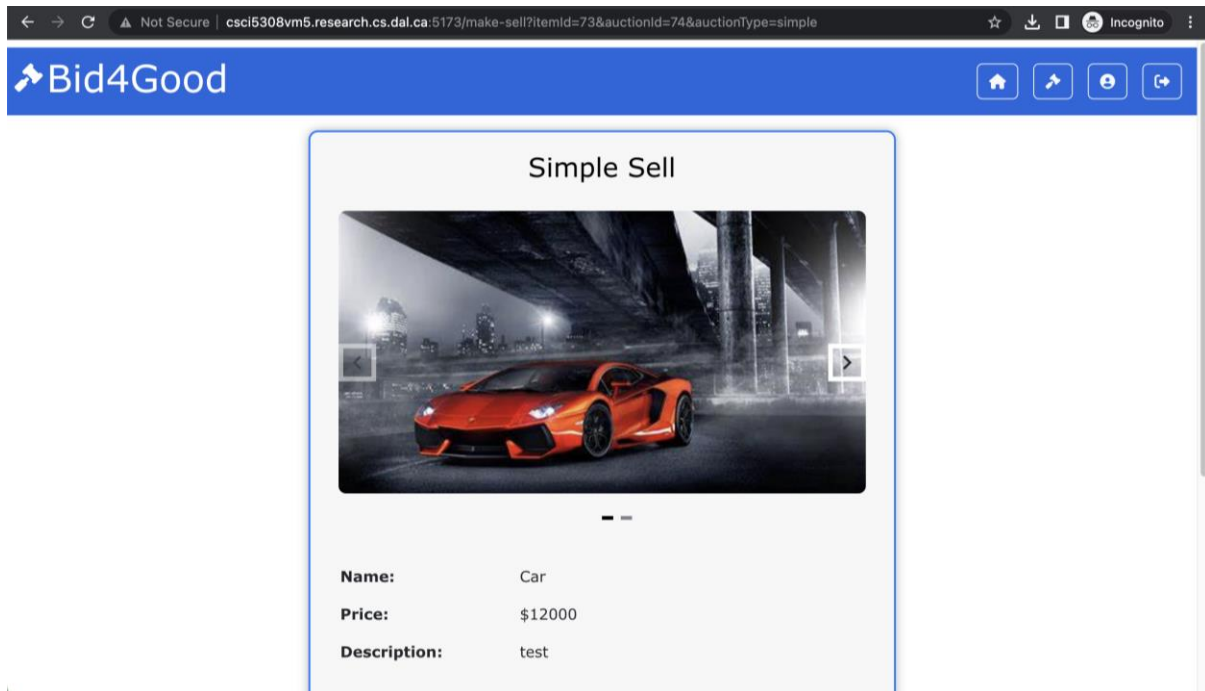
Post-conditions:

- The user can view the results of the blind auction and complete the payment for the product.

Exceptions:

- If the user's bid amount is not higher than the current highest bid, the platform displays an error message and prompts them to enter a higher bid amount.
- If the user does not have sufficient funds or payment methods, the platform displays an error message and prompts the user to add funds or payment methods.
- If there are technical issues or server errors, the platform displays an error message and prompts the user to try again later.

Use Case: Buying Items in Normal Sell



Primary Actor: User

Stakeholders:

- User: The person who wants to buy an item on the platform.
- Platform: The system that provides normal sell functionality.

Preconditions:

- The user is logged in to the platform.
- The user has an item they want to sell.
- The platform supports normal sell functionality.

Main Success Scenario:

1. The user navigates to the normal sell page of the platform.
2. The platform displays a list of available items for sale.
3. The user clicks on an item they want to buy.
4. The platform displays more details about the item, such as the name, description, price, and image.
5. The user clicks on the "Buy" button.
6. The platform redirects the user to a payment page where they can enter payment details and complete the transaction.
7. The platform updates the item status to "sold" and notifies the seller of the purchase.

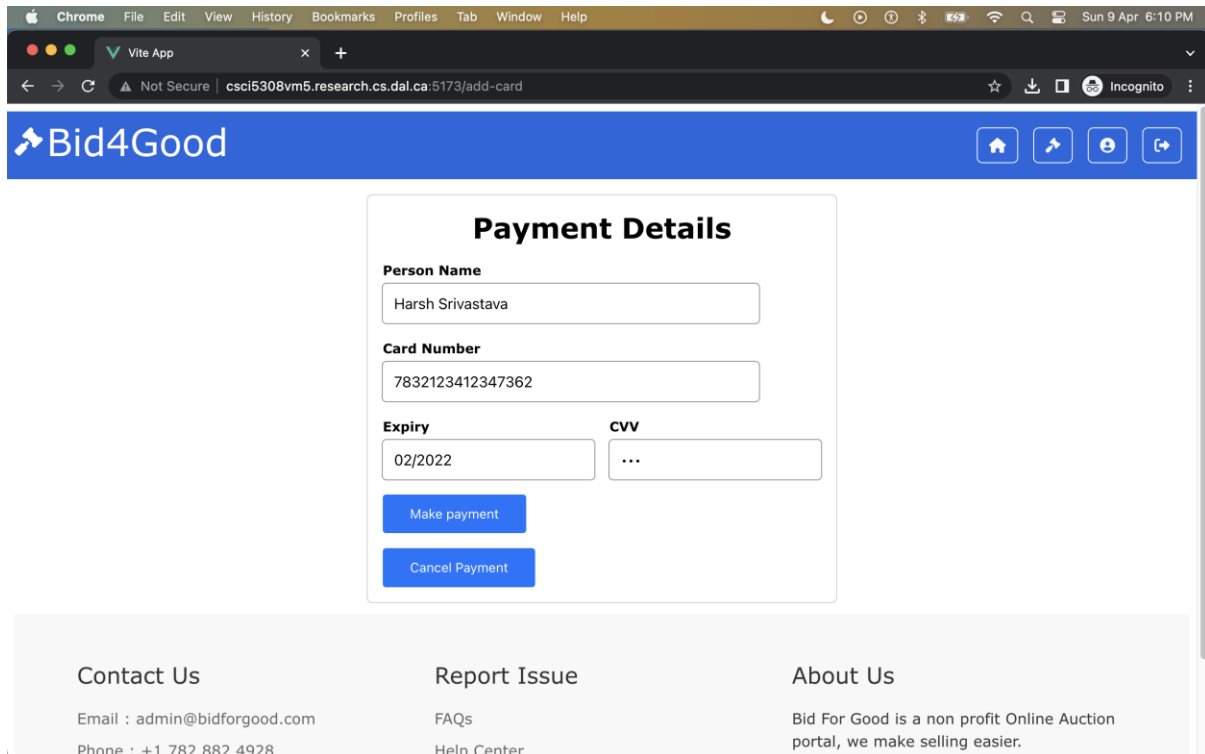
Post-conditions:

- The user can sell or buy items on the platform.

Exceptions:

- If the user enters invalid information, the platform displays an error message and prompts them to correct the input.
- If the user tries to sell an item with incomplete or missing information, the platform displays an error message and prompts them to fill in all required fields.
- If the user experiences technical issues or server errors during the transaction, the platform displays an error message and prompts the user to try again later.

Use Case: Making Payment and Placing Order

A screenshot of a web browser showing the Bid4Good payment page. The browser's address bar shows a URL starting with 'csci5308vm5.research.cs.dal.ca:5173/add-card'. The page has a blue header with the Bid4Good logo and navigation icons. The main content area is titled 'Payment Details' and contains several input fields: 'Person Name' (filled with 'Harsh Srivastava'), 'Card Number' (filled with '7832123412347362'), 'Expiry' (filled with '02/2022'), and 'CVV' (filled with '...'). Below these fields are two buttons: 'Make payment' and 'Cancel Payment'. At the bottom, there is a footer section with three columns: 'Contact Us' (with email 'admin@bidforgood.com' and phone '+1 782 882 4928'), 'Report Issue' (with links for 'FAQs' and 'Help Center'), and 'About Us' (with a description of Bid For Good as a non-profit online auction portal).

Primary Actor: User

Stakeholders:

- User: The person who wants to make a payment and place an order.

Preconditions:

- The user has selected the payment option and navigated to the payment page.
- The user has a valid payment card and is ready to provide payment information.

Main Success Scenario:

1. The user is presented with a payment page containing fields for payment information, including card number, expiration date, and security code.
2. The user enters their payment information into the fields provided.
3. The user confirms the payment details and clicks the "Pay" button.
4. The payment gateway processes the payment transaction and verifies the payment information.
5. The order is placed and user is redirected to Homepage.

Post-conditions:

- The payment is successfully processed, and the user's order is placed.

Exceptions:

- If the user enters invalid payment information, the payment gateway displays an error message and prompts the user to correct the input.
- If the payment gateway is unable to process the payment transaction, the user is presented with an error message and prompted to try again later.

Use Case: Editing User Account Details

Vite App x +

Not Secure | csci5308vm5.research.cs.dal.ca:5173/buyer

Bid4Good

Details

Report an Issue

Issue History

User Details

Hello: checkSeller

First Name: checkSeller

Last Name: only

Phone Number: 1234567890

Date of birth: 05/04/2023

Address: test

State: Alberta

City: Airdrie

Postal Zip Code: a1b-2c3

format: a1b-2c3 | a1b2c3 | a1b 2c3

User Credentials

Email: checkselleronly@dal.ca

Once Registered Can't be Changed

Role: Contact BidForGood

☐ Buyer

Primary Actor: User

Stakeholders:

- User: The person who wants to edit their account details.
- Platform: The system that provides user account functionality.

Preconditions:

- The user is registered and logged in to the platform.
- The user has access to the account settings page.
- The user wants to update their account details such as name, address, phone, and other details.

Main Success Scenario:

1. The user navigates to the account settings page and selects the option to edit their account details.
2. The platform presents a form containing fields for the user to update their account details, such as name, address, phone, and other details.
3. The user enters the updated account details into the form.
4. The user confirms the updated account details and clicks the "Save Changes" button.
5. The platform validates the updated account details and ensures that all required fields are filled.

6. The platform saves the updated account details in the system and displays a success message to the user.
7. The user can now view their updated account details on the platform.

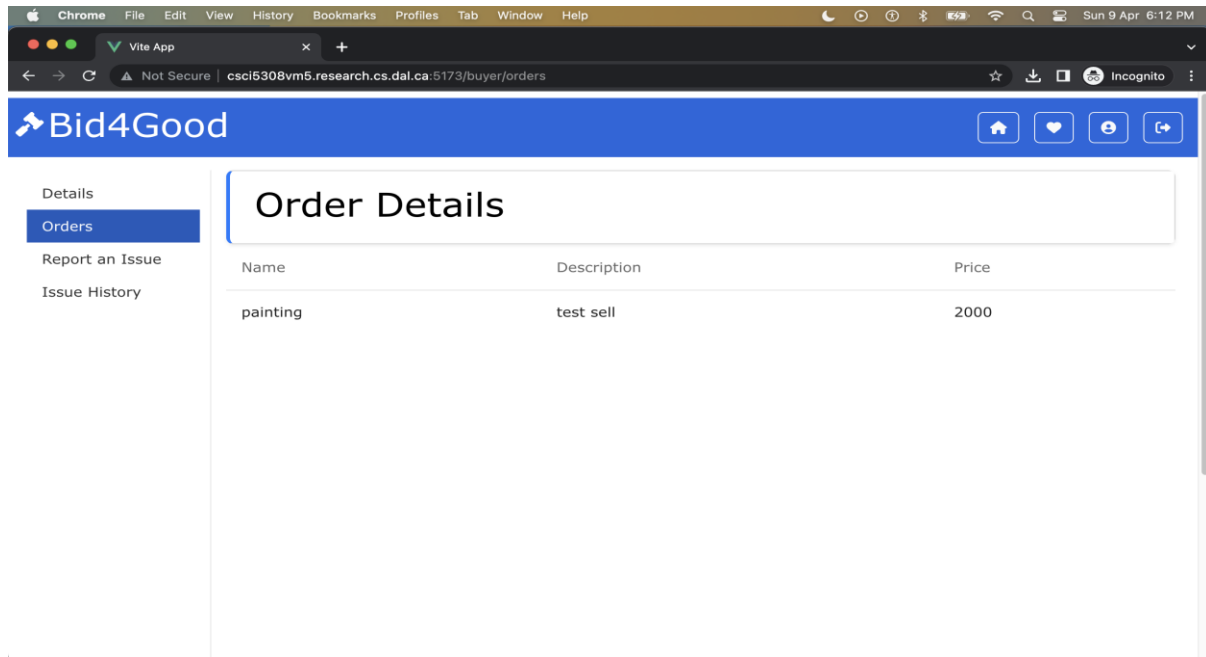
Post-conditions:

- The user's account details are successfully updated in the system.
- The user can view their updated account details on the platform.

Exceptions:

- If the user enters invalid account details, the platform displays an error message and prompts them to correct the input.
- If the user tries to update their account details with incomplete or missing information, the platform displays an error message and prompts them to fill in all required fields.

Use Case: Viewing Order History



Primary Actor: Buyer

Stakeholders:

- User: The person who wants to view their order history.
- Platform: The system that provides the order history functionality.

Preconditions:

- The user is registered and logged in to the platform.
- The user has placed at least one order on the platform.

Main Success Scenario:

1. The user navigates to the "Order History" page on the platform.
2. The platform displays a list of the user's orders, including the order number, Item name and the price paid.
3. The user can navigate back to the order list or return to the platform's home page.

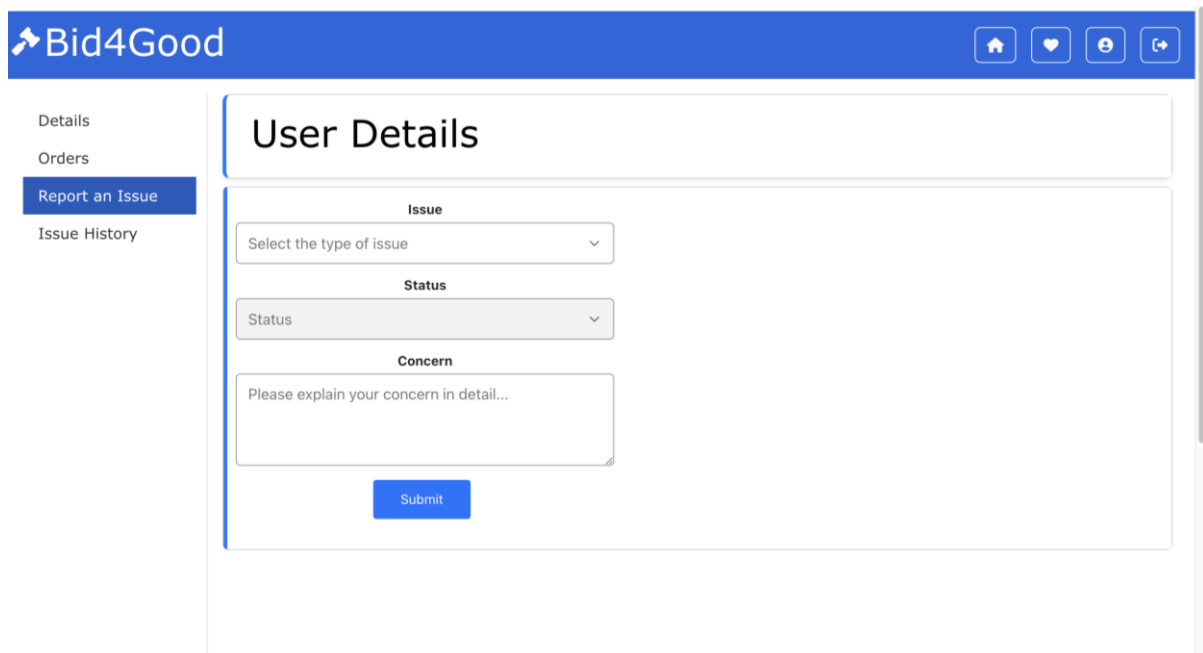
Post-conditions:

- The user has viewed their order history on the platform.
- The user can navigate back to the order list or return to the platform's home page.

Exceptions:

- If the user has not placed any orders on the platform, the order history page displays a message informing the user that they have no order history.
- If the platform encounters an error while retrieving the user's order history, the order history page displays an error message and prompts the user to try again later.

Use Case: Reporting an Issue



Bid4Good

Details
Orders
Report an Issue
Issue History

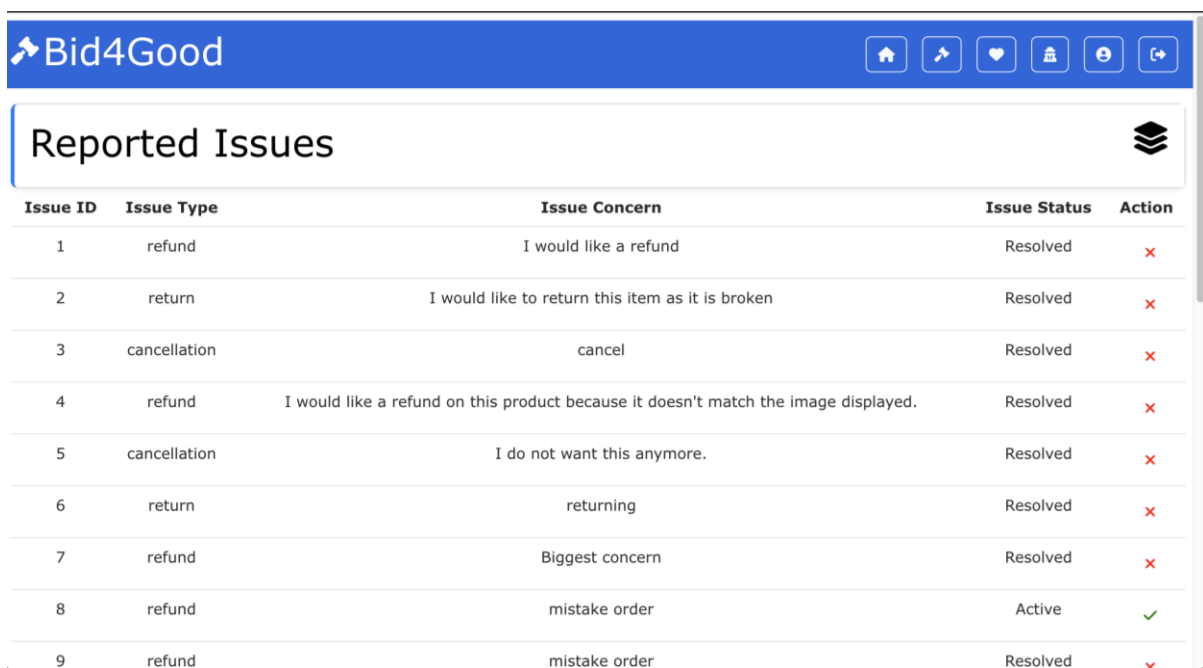
User Details

Issue
Select the type of issue

Status
Status

Concern
Please explain your concern in detail...

Submit



Bid4Good

Reported Issues

Issue ID	Issue Type	Issue Concern	Issue Status	Action
1	refund	I would like a refund	Resolved	✗
2	return	I would like to return this item as it is broken	Resolved	✗
3	cancellation	cancel	Resolved	✗
4	refund	I would like a refund on this product because it doesn't match the image displayed.	Resolved	✗
5	cancellation	I do not want this anymore.	Resolved	✗
6	return	returning	Resolved	✗
7	refund	Biggest concern	Resolved	✗
8	refund	mistake order	Active	✓
9	refund	mistake order	Resolved	✗

Primary Actor: User

Stakeholders:

- User: The person who reports the issue related to refund, return, or cancellation.
- Admin: The support staff who receives and resolves the issue ticket.

Preconditions:

- The user has encountered an issue related to refund, return, or cancellation and wants to report it.
- The user is logged in to the platform.

Main Success Scenario:

1. The user navigates to the "Report Issue" section of the platform.
2. The user selects the type of issue (refund, return, or cancellation) and describes the problem in detail.
3. The user submits the issue report.
4. The platform creates an active issue ticket and sends it to the admin.
5. The admin receives the issue ticket and begins working on a resolution.
6. The admin resolves the issue and marks the ticket as resolved.
7. The platform updates the issue status to resolved and archives it in the report history.

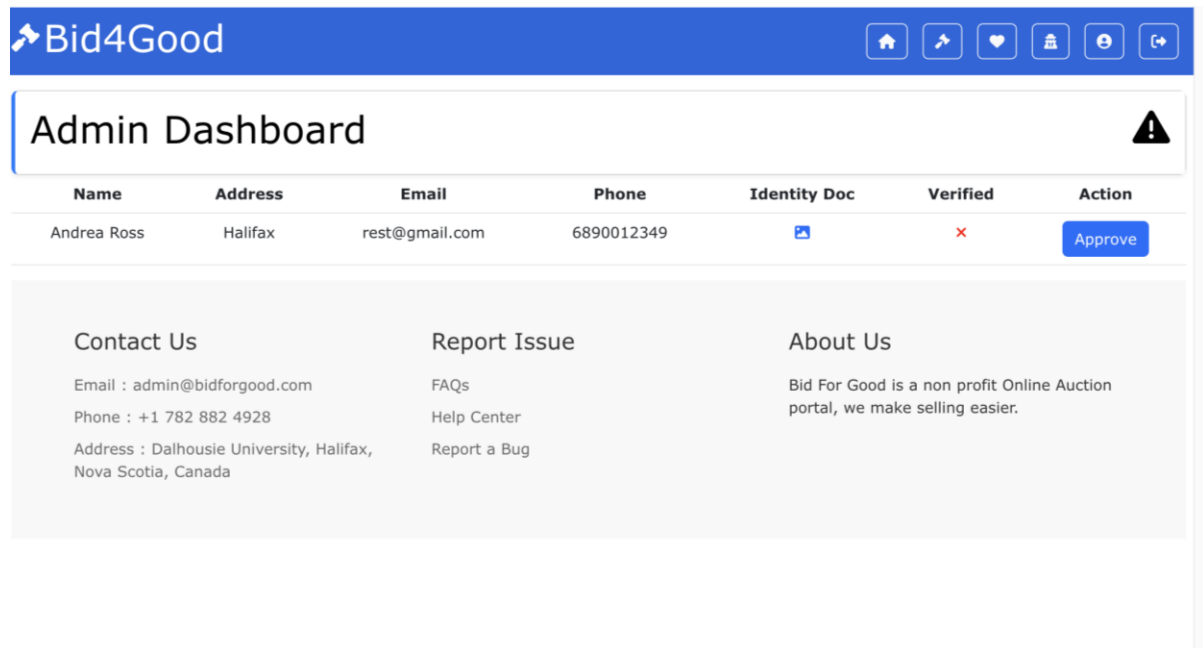
Post-conditions:

- The user's issue is resolved, and the platform updates the issue status to resolved.
- The admin has resolved the issue and marked the ticket as resolved.
- The resolved issue is archived in the report history for future reference.

Exceptions:

- If the user submits incomplete or invalid information, the platform displays an error message and prompts the user to correct the input.
- If the admin is unable to resolve the issue, the platform updates the issue status to unresolved and sends a notification to the user.
- If the user does not receive a resolution to their issue within a reasonable timeframe, the user may escalate the issue to a higher level of support.

Use Case: Admin Dashboard



Primary Actor: Admin

Stakeholders:

- Admin: The person who manages the platform's backend and performs administrative tasks.
- User: The person who uses the platform and interacts with the frontend.

Preconditions:

- The admin is logged in to the platform's backend.
- The admin has the necessary permissions to access the admin dashboard.

Main Success Scenario:

1. The admin navigates to the admin dashboard.
2. The admin sees a list of pending user verifications and can approve or reject them.
3. The admin sees a list of active report tickets and can resolve them.
4. The admin can access any page on the platform without any limitations or restrictions.

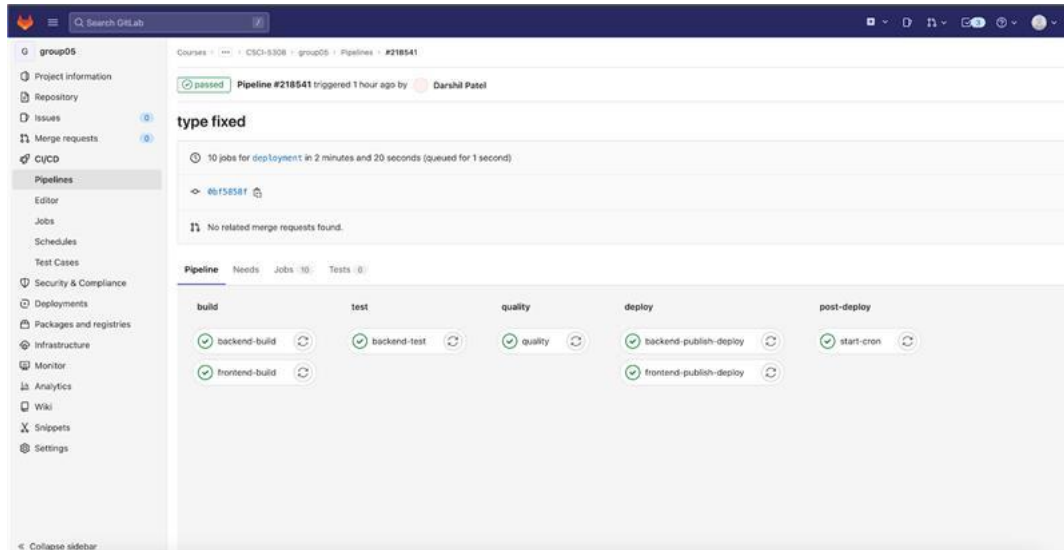
Post-conditions:

- The admin has reviewed and approved/rejected user verification requests.
- The admin has resolved any active report tickets.
- The admin can access any page on the platform without any limitations or restrictions.

Exceptions:

- If the admin encounters an error or issue while performing administrative tasks, the platform displays an error message and prompts the admin to resolve the issue or contact technical support.
- If the admin attempts to perform an unauthorized action or access a restricted page, the platform displays an error message and denies access.

Continuous Integration/Continuous Deployment



The CI/CD pipeline automates the process of building, testing, and deploying software and is composed of five stages: **Build, Test, Quality, Deploy, and Post-Deploy**.

The CI section, comprising the Build, Test, and Quality stages, is responsible for constructing and testing the software to ensure that it meets quality requirements. To build both the frontend and backend, the "**npm run build**" command is used in the Build stage, resulting in a build package for each. The Test stage verifies all test cases, and if they pass, the Quality stage follows. If any test fails, the pipeline fails. The "**npm run test**" command is used to execute the tests.

In the Quality stage, ESLint detects code quality issues like commented code, unused imports, and missing "id" attributes in tables. If there are no issues, a JSON artifact containing all of the linted code is created and stored in the corresponding pipeline within the CI/CD pipelines. After completing the CI pipeline, the CD pipeline commences, which involves the Deploy and Post-Deploy phases, which manage the software's deployment to the VM. During the Deploy stage, the built modules are sent to the VM.

The Post-Deploy phase includes running a cron job at fixed intervals to re-run the code. This is necessary to send mails to the winners. The CI/CD pipeline speeds up and makes software updates more reliable by automating these stages.

Test Driven Development

The BidForGood application was tested using the following dependencies. In total, 60 tests were written, consisting of both unit and integration tests.

- **mocha**: A JavaScript test framework.
- **nyc**: A command-line interface for Istanbul, a code coverage tool.
- **chai**: A BDD/TDD assertion library for Node.js and the browser.
- **chai-as-promised**: A plugin for Chai that provides assertions for Promises.
- **supertest**: A library that allows you to test HTTP calls made by your Express app.
- **sinon**: A library for testing JavaScript code that relies on functions outside of a unit.
- **sinon-express-mock**: A plugin for Sinon that provides mocks and stubs for Express.js.

Instructions to run Test cases:

1. Go to the Server directory by running the command: `cd /server/`.
2. Install the required dependencies if not installed by running the command: `npm install`.
3. Compile the TypeScript code into JavaScript by running the command: `tsc -w`.
4. Run Test cases by running the command: `npm run test`.

Code Coverage Report:

60 passing (4s)

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	89.75	50	84.89	89.65	
controllers	76.95	54.34	70.58	76.27	
addBidItems.ts	78.57	100	42.85	77.77	95-101,107-113
adminCont.ts	95.23	80	100	95.23	58
fetchBidDetails.ts	100	100	100	100	
fetchBidItems.ts	100	100	100	100	
fetchDetails.ts	100	75	100	100	41
orderItemCont.ts	72.72	100	50	70	34-53
placeBid.ts	92.3	50	100	92.3	21
registerCont.ts	46.25	20	60	45.56	...158-194, 214, 220-259, 297
reportCont.ts	95.23	83.33	100	95.23	46-47
wishlistCont.ts	80	100	66.66	77.77	27-41
controllers/liveAuctionController	33.87	0	9.52	33.87	
auctionHandler.ts	33.87	0	9.52	33.87	...138,143-154,159,163-184
models	100	100	100	100	
aunctionModel.ts	100	100	100	100	
imageDetails.ts	100	100	100	100	
itemModel.ts	100	100	100	100	
loginDetailModel.ts	100	100	100	100	
orderDetailsModel.ts	100	100	100	100	
reportsModel.ts	100	100	100	100	
userBidDetails.ts	100	100	100	100	
userDetailModel.ts	100	100	100	100	
wishlistModel.ts	100	100	100	100	
tests	100	100	100	100	
IntegrationLoginLogoutTest.spec.ts	100	100	100	100	
IntegrationPasswordHashingTest.spec.ts	100	100	100	100	
IntegrationTest.spec.ts	100	100	100	100	
IntegrationEmailSenderTest.spec.ts	100	100	100	100	
addBidItemsTest.spec.ts	100	100	100	100	
adminTest.spec.ts	100	100	100	100	
auctionHandlerTest.spec.ts	100	100	100	100	
emailSenderIntegrationTest.spec.ts	100	100	100	100	
fetchBidDetailsTest.spec.ts	100	100	100	100	
fetchBidItemTest.spec.ts	100	100	100	100	
fetchDetailsTest.spec.ts	100	100	100	100	
integrationRegisterTest.spec.ts	100	100	100	100	
loginIntegrationTest.spec.ts	100	100	100	100	
orderItemTest.spec.ts	100	100	100	100	
placeBids.spec.ts	100	100	100	100	
registerIntegrationTest.spec.ts	100	100	100	100	
registerTest.spec.ts	100	100	100	100	
reportTest.spec.ts	100	100	100	100	
wishlistTest.spec.ts	100	100	100	100	
util	96.29	100	100	96	
constants.ts	100	100	100	100	
database.ts	100	100	100	100	
emailSender.ts	87.5	100	100	87.5	39
passwordHashing.ts	100	100	100	100	
path.ts	100	100	100	100	

Reconnect to Discord Gateway No LOC

Go Live

At present, our application has a code coverage of 89.75%. However, there are some files and methods, such as the register controller and auction controller, which have lower code coverage due to their complex functionality and interactions with the database. Consequently, writing test cases for these components was particularly challenging.

In addition to the aforementioned challenges, our team utilized various techniques to eliminate code smells from the test cases. One such technique was the implementation of **stubs** and **mocks**. These techniques enabled us to isolate and test individual components of the application more effectively, ultimately improving the overall quality of the test suite.

Code Smells and Code Quality

Initial Stage

Smells detected:

During the initial run, 172 smells were detected which included a majority of commented code, unused imports and importing multiple entities from one library. Only two methods had implementation smell for cognitive complexity.

Cognitive compexity smells(implementation) at:	Status	Reason
FrontEnd/src/router/index.ts	Unresolved	It could not be removed because it provides different access to different users
server/dev/controllers/generateWinner.ts	Unresolved	It could not be removed because changing anything would break the feature.

The following tools have been used to analysing the code quality and/or smells in the code.

- **Sonarcloud**
- **ESlint**
- **Sonarlint**

Each API in the controller folder has a single responsibility, hence adhering to the SOLID design principle. The code follows the Model-View-Controller (MVC) design paradigm. These patterns promote cohesiveness by splitting issues into discrete layers and, as a result, diminish coupling by encouraging loose coupling between the layers.

Models define the structure of the data stored in the database, views describe the displays, and controllers perform logical conditioning and get data from the database.

To discover issues and code smells in the code, we utilized Sonarlint and SonarCloud. The softwares did not detect architecture or implementation smells. Minor smells were reported by the software. These include unnecessary imports, commented code, and unused/renamed variables. All of these issues have been fixed and are included in the code.

We followed clean code guidelines, such as providing comments to each function. The functions have been kept basic and modularized to reduce code complexity and hence adhere to coding standards. The only issues that remain unresolved are in the generateWinner and router files. These issues have resulted due to the complicated conditioning. They have not been resolved due to functionality requirements and multiple checks. Also, to avoid unintentional code breakage we have not tried not to change.

Code Smells:

SonarLint and SonarCloud found code smells in the code, which included commented out code in the files. They have been fully erased from the files. There were further issues like the unexpected use of the 'var' type rather than 'const' or 'let'. These problems have also been resolved. A couple of worrying concerns revealed by the softwares included the direct exposing of aws-keys and passwords in the field. Because the users' identity cards are stored, this could have resulted in a breach of their privacy. To avoid this, we've put the keys in the .env files. There were some redundant imports in the files that were also eliminated.

The Link to the excel sheet: -

<https://docs.google.com/spreadsheets/d/1MXqItm1xL9N7h9cg4Zay7WNO53PQor4900yII6kuGiM/edit#gid=968508203>

The screenshot displays the SonarCloud web interface for a project named 'Bid4Good'. The 'Issues' tab is active, showing a list of security vulnerabilities. On the left, a 'Filters' sidebar shows the following counts: 0 Bugs, 7 Vulnerabilities, 0 Code Smells, 7 Blockers, 113 Minors, 3 Criticals, and 0 Infos. The main panel lists three vulnerabilities, all categorized as 'Vulnerability' and 'Blocker'.

Issue Title	Severity	Open	Blocked	Not assigned	Effort	Age
Make sure this AWS Secret Access Key is not disclosed.	Vulnerability	Open	Blocked	Not assigned	30min effort	last month
Change this code to not construct SQL queries directly from user-controlled data.	Vulnerability	Open	Blocked	Not assigned	30min effort	13 days ago
Make sure this AWS Access Key ID is not disclosed.	Vulnerability	Open	Blocked	Not assigned	30min effort	last month
Make sure this AWS Secret Access Key is not disclosed.	Vulnerability	Open	Blocked	Not assigned	30min effort	last month

The bottom of the list indicates '7 of 7 shown'.

My Projects My Issues Explore Q

Bid4Good > ProjectASDC > master

Summary Issues Security Hotspots Measures Code Activity

Filters [Clear All Filters](#)

Type

- Bug 0
- Vulnerability 0
- Code Smell 3

Severity

- Blocker 7
- Minor 113
- Critical 3**
- Info 0
- Major 86

Add to selection **Ctrl + click**

FrontEnd/src/router/index.ts

- ☐ [Refactor this function to reduce its Cognitive Complexity from 25 to the 15 allowed.](#) No tags +
Code Smell Open Critical Not assigned 15min effort · 21 hours ago

server/dev/controllers/fetchBidDetails.ts

- ☐ [Unexpected var, use let or const instead.](#) No tags +
Code Smell Open Critical Not assigned 5min effort · 28 days ago

server/dev/controllers/generateWinner.ts

- ☐ [Refactor this function to reduce its Cognitive Complexity from 21 to the 15 allowed.](#) No tags +
Code Smell Open Critical Not assigned 11min effort · 22 hours ago

3 of 3 shown

My Projects My Issues Explore Q

Bid4Good > ProjectASDC > master

Summary Issues Security Hotspots Measures Code Activity

Filters [Clear All Filters](#)

Type

- Bug 9
- Vulnerability 0
- Code Smell 104**

Severity

- Blocker 0
- Minor 104**
- Critical 3
- Info 0
- Major 65

Add to selection **Ctrl + click**

FrontEnd/src/components/admin-dashboard.vue

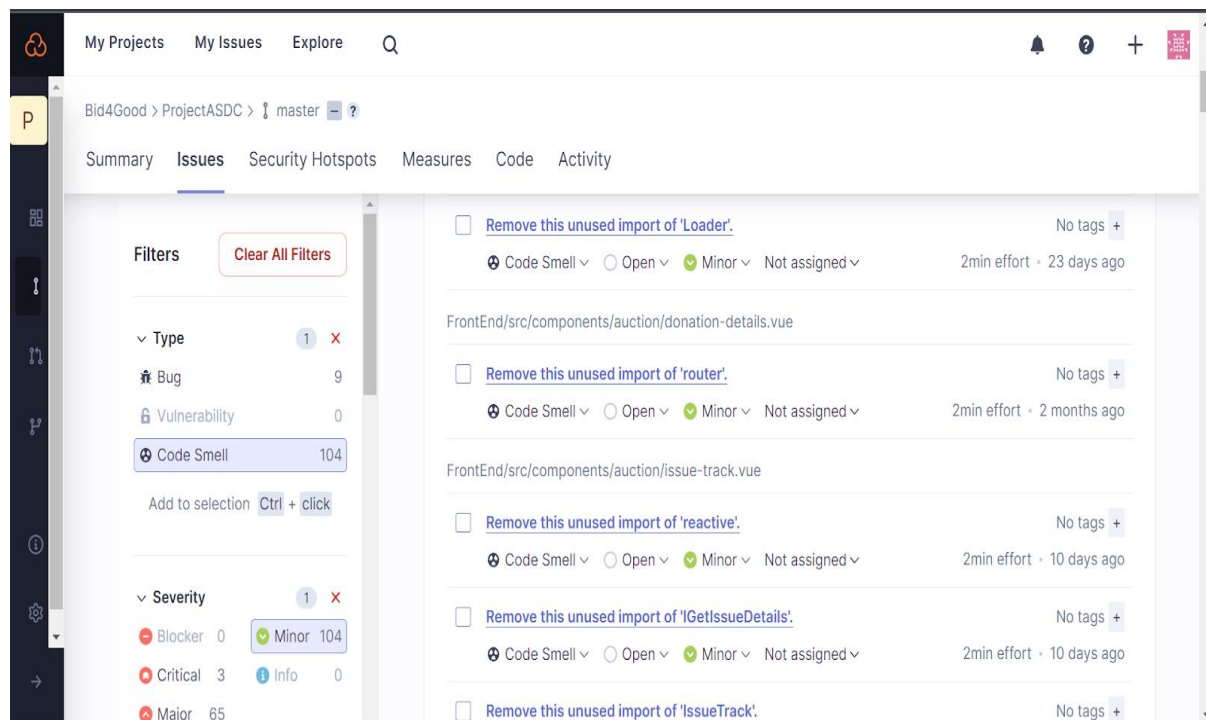
- ☐ [Remove this unused import of 'NoContent'.](#) No tags +
Code Smell Open Minor Not assigned 2min effort · 27 days ago

FrontEnd/src/components/admin-issue-details.vue

- ☐ [Remove this unused import of 'NoContent'.](#) No tags +
Code Smell Open Minor Not assigned 2min effort · 10 days ago
- ☐ [Replace this "String" wrapper object with primitive type "string".](#) No tags +
Code Smell Open Minor Not assigned 1min effort · 10 days ago

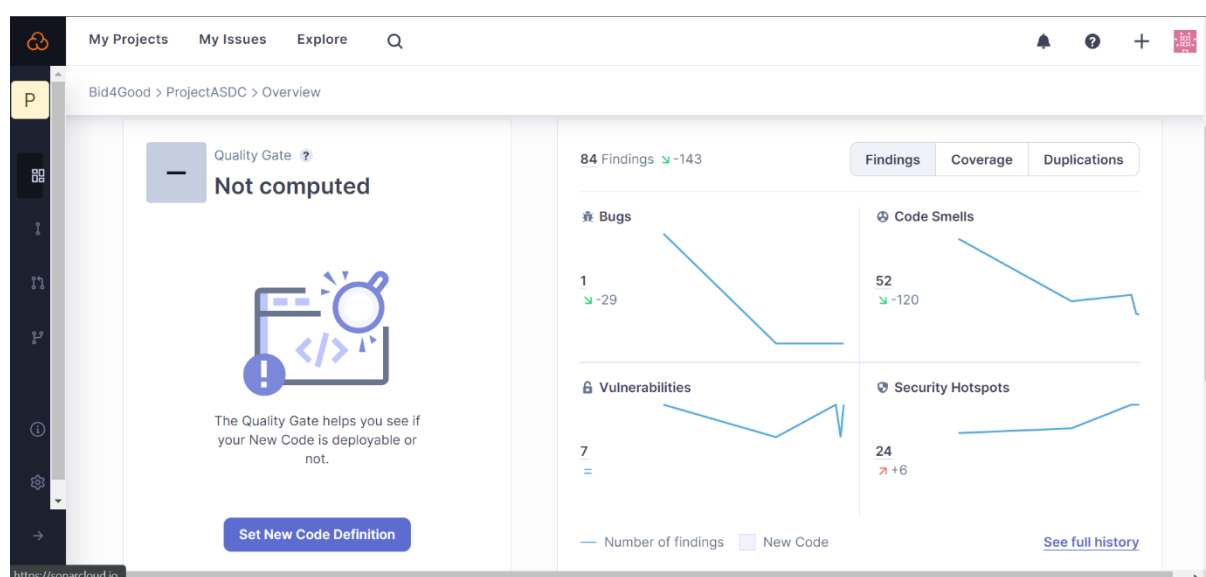
FrontEnd/src/components/auction/blind-auction.vue

- ☐ [Remove this unused import of 'LocationQueryValue'.](#) No tags +
Code Smell Open Minor Not assigned 2min effort · 10 days ago



Current Stage:

We followed clean code guidelines, such as providing comments to each function. The functions have been kept basic and modularized to reduce code complexity and hence adhere to coding standards. The only issues that remain unresolved are in the generateWinner, integrationregisterTest and router files. These issues have resulted due to the complicated conditioning and internal dependency. They have not been resolved due to functionality requirements and multiple checks. Also, to avoid unintentional code breakage we have not tried not to change.



The screenshot displays the SonarQube web interface for a project named 'ProjectASDC'. The interface is divided into a left sidebar for filters and a main panel for issue details.

Filters:

- Type:** Bug (0), Vulnerability (0), Code Smell (5). A 'Clear All Filters' button is present.
- Severity:** Blocker (0), Minor (26), Critical (5), Info (0), Major (21).

Main Panel:

- Top Bar:** Includes a 'Bulk Change' button, 'Select issues' dropdown, 'Navigate to issue' controls, and a summary of '1 / 5 issues' with a '29min effort'.
- Issue 1:** Located at 'FrontEnd/src/router/index.ts'. The issue is 'Refactor this function to reduce its Cognitive Complexity from 25 to the 15 allowed.' It is a 'Code Smell' with 'Critical' severity, 'Not assigned', and '15min effort - 1 hour ago'.
- Issue 2:** Located at 'server/dev/controllers/generateWinner.ts'. The issue is 'Refactor this function to reduce its Cognitive Complexity from 21 to the 15 allowed.' It is a 'Code Smell' with 'Critical' severity, 'Not assigned', and '11min effort - yesterday'.
- Issue 3:** Located at 'server/dev/tests/integrationRegisterTest.spec.ts'. The issue is 'Refactor this redundant 'await' on a non-promise.' It is a 'Code Smell' with 'Critical' severity, 'Not assigned', and '1min effort - 4 hours ago'.

Project Accomplishments

The Auction Website Project was successfully completed within the stipulated timeframe and budget. The project team achieved the following key accomplishments:

- Developed a responsive and user-friendly website design.
- Created a backend system using Node.js and Express that could handle multiple concurrent connections.
- Integrated Vue.js to develop a dynamic and responsive frontend with real-time updates using live sockets.
- Incorporated SQL databases for reliable and efficient data storage and retrieval.
- Developed a robust admin system to verify seller IDs.
- Implemented features for live and blind auctions, as well as a marketplace for verified sellers to sell their items.
- Ensured the website was secure and user-friendly for both buyers and sellers.

Challenges Faced:

During the project, the team faced some challenges, such as:

- Integration issues between different technologies used.
- Implementing and testing the admin system for verifying seller IDs.
- Ensuring the website was user-friendly and met the requirements of both buyers and sellers.

Lessons Learned:

The project team gained valuable experience and knowledge from this project, including:

- Working with new technologies and tools such as Node.js, Express, Vue.js, SQL databases, and live sockets.
- Integrating frontend and backend systems to create a seamless user experience.
- Implementing and testing security measures to protect sensitive data.
- Collaborating effectively within a team to achieve project goals.

Conclusion: The Auction Website Project was successfully completed, meeting all project requirements and objectives. The project team was able to deliver a user-friendly and secure website that enabled verified sellers to create auctions and sell items, while allowing buyers to participate in auctions and make purchases. The experience gained from this project will serve as a valuable asset for future development projects.